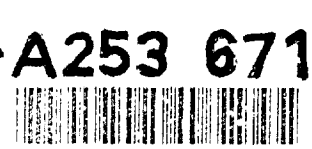


(2)

NWC TP 7207

Computational Fluid Dynamics Capability for Internally Carried Store Separation

by

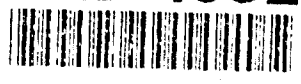
Sukumar R. Chakravarthy
Kuo-Yen Szema
Computational Fluid Dynamics Department
Rockwell International Science Center

for
Attack Weapons Department

DECEMBER 1991

NAVAL WEAPONS CENTER
CHINA LAKE, CA 93555-6001

Approved for public release;
distribution is unlimited



Naval Weapons Center

FOREWORD

This report documents a program having as its objective the conduct of an algorithm research and development effort that would result in the demonstrated capability to provide accurate numerical flowfield data for situations typical of weapon carriage and release from an internal weapon bay. This research was sponsored by the Naval Weapons Center (NWC), China Lake, Calif.

This work was performed by Rockwell International Science Center (RISC) under Contract No. N60530-90-C-0393 during the period from 26 September 1990 to 26 September 1991. Technical direction was provided by Dr. Robert Burman of the Naval Weapons Center. Funding for this effort was provided by the airframes technology portion of the Air Launched Weaponry Technology Block Program at the Naval Weapons Center under the sponsorship of the AAW/SSUW/SAT Division of the Office of Naval Technology (OCNR-213).

The principal investigators at RISC were Sukumar R. Chakravarthy and Kuo-Yen Szema. Other personnel of the RISC Computational Fluid Dynamics Department supported the effort.

In the interest of cost savings, the document prepared by RISC was used without change for this NWC technical publication. Consequently, there are deviations from standard NWC style and format.

The report was reviewed for technical accuracy at NWC by Dr. Burman. It is released for information only. The views and conclusions are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Naval Weapons Center.

Approved by
P. B. HOMER, *Head*
Attack Weapons Department
1 December 1991

Under authority of
D. W. COOK
Capt., U.S. Navy
Commander

Released for publication by
W. B. PORTER
Technical Director

NWC Technical Publication 7207

Published by.....Technical Information Department
Collation.....Cover, 92 leaves
First printing.....170 copies

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1991	Final; 9/26/90 to 92691

4. TITLE AND SUBTITLE

COMPUTATIONAL FLUID DYNAMICS CAPABILITY FOR INTERNALLY CARRIED STORE SEPARATION

5. FUNDING NUMBERS

Contract N60530-90-C-0393

6. AUTHOR(S)

Chakravarthy, Sukumar R., and Szema, Kuo-Yen

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Rockwell International Science Center
Thousand Oaks, CA 91358

8. PERFORMING ORGANIZATION REPORT NUMBER

SC71039TR

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Naval Weapons Center
China Lake, CA 93555-6001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

NWC TP 7207

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

A Statement: Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

(U) The ability to predict safe and effective release of air-launched weapons carried internally in a weapon bay or externally on a pylon/rack is required by the Safety Committee before approval is given for a flight test to validate various store separation characteristics. Because of the novel conditions attendant to test and evaluation (T&E) testing, only limited experimental and flight data are available beforehand to aid in assessing safe weapon release. Computational methods have the potential of resolving, in a timely manner, the store separation problem without these weaknesses. Two areas need to be addressed. First, an accurate computational algorithm is needed to minimize numerical uncertainty as the store is tracked in time. Second, correct physical models must be implemented to model the moving store, including grid adaptation. The general objective of this program is to conduct an algorithm research and development effort that will result in demonstrating the capability to provide accurate numerical flowfield predictions for situation of weapon carriage and release from an internal weapons bay. Based on a truly multidimensional unrestricted Essentially Non-oscillatory (ENO) scheme, a new UNIVERSE (Unification of essentially Non-oscillatory Interpolation techniques with a geometrically VERSatilE) series code has been developed at the Rockwell International Science Center. The new ENO concepts are state-of-the-art interpolation schemes that work with arbitrary cell shapes. For example, hexahedral, triangular prism, and tetrahedral elements (conservation cells) can all be covered in a unified manner. This also implies that both structured and unstructured bookkeeping schemes can be employed to conveniently treat complex topologies. A new suite of fully automatic flexible-cell mesh generation methods is also being developed for integration with the new series

(U) This is the final report draft for Phase I (contract awarded in September 1990 by the Naval Weapons Center to Rockwell International Science Center). The report is organized into two major sections: Section I is a synopsis of Phase I work, and Section II contains all the technical details. Numerical results are presented for the linear wave equation, Burgers' equation, two-dimensional "aircraft" and "store," F-18 configuration, Space Shuttle multibody configuration, and other cases. Solutions are in very good agreement with available exact solutions and experimental data.

14. SUBJECT TERMS

Fluid Dynamics, Computational Fluid Dynamics, Riemann Problem, Unstructured Grid, Geometry Formulation

15. NUMBER OF PAGES

182

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

CONTENTS

SECTION I
SYNOPSIS

1.0 INTRODUCTION

The design of modern aerospace vehicle concepts increasingly requires the detailed understanding and synergistic use of ever more complex fluid phenomena. Both industry and government view Computational Fluid Dynamics (CFD) as a critical, potentially efficient and cost-effective technology for the development of advanced aerospace configurations. The overall technical problem in CFD is to devise reliable numerical approaches to simulate the complex fluid physics arising in flow about complex shapes/topologies. Any approach to the solution of this problem must address the following issues: (a) appropriate modeling of the physics, (b) proper match between physics and numerics, (c) accuracy, (d) geometric flexibility, and (e) ease of use. In today's environment, these issues must also be treated in ways which are compatible and synergistic with both proven and emerging high-performance computer architectures. The major thrust of the Science Center's CFD program is to effect rapid but sustained strides in algorithm and code development to exploit the increasing power of supercomputers, super-minicomputers, parallel computers, and graphics workstations in order to systematically and continually improve the use of CFD in state-of-the-art design and analysis scenarios.

In FY 90, we embarked on an ambitious plan to begin the development of the next generation of CFD codes and technology — the successors to the proven and successful USA-series (Unified Solution Algorithms) of codes that are currently in widespread use in Rockwell and in Government agencies. The new generation is based on very significant developments in computational algorithms accomplished in past and present research and will also benefit from the considerable experience the CFD Department's researchers have acquired in working with more than one hundred users throughout Rockwell and in other DoD agencies.

In this endeavor, we looked forward to participate in Government-sponsored research programs which are particularly well matched with our quest for the next higher plateau in CFD capability. The current contract (awarded September, 1990 by NWC to Rockwell International Science Center) fits that category extremely well. The general objective of this program is to conduct an algorithm research and development effort that will result in the demonstrated capability to provide accurate

numerical flowfield data for situations typical of weapon carriage and release from an internal weapon bay. This program permits us to apply and specialize the very general algorithmic capability that we have developed to the challenging problem of computing the airloads and separation characteristics of internally carried weapons. It will serve as a key demonstration of the flexibility of the new capability that we have built. In turn, we hope to provide NWC with a very powerful new CFD capability that will serve them well for years to come. During the first phase of the program, the goal is to demonstrate inviscid 3-d static and 2-d dynamic capability. In the next phase, the focus will be on viscous static 3-d and inviscid dynamic 3-d capability. The third phase deals with 3-d viscous and dynamic CFD analysis of a missile emerging from a weapon bay for which the missile trajectory is determined by integrating the CFD flowfield solver with a six degree-of-freedom trajectory algorithm. This report presents work done during Phase I along with detailed reference material for the overall computational framework.

This report is organized into two major sections identified by Roman numerals: Section I is a synopsis of Phase I work and Section II contains all the technical details. These major sections include several chapters each. For convenience, these are identified as sections with Arabic numerical identifiers. Section 2 presents a summary of work performed during Phase I followed by Section 3 containing conclusions that can be drawn at this point during the three-phase program. Section 4 discusses the overall numerical framework for the current generation of UNIVERSE-series codes and forms the background for all subsequent sections. Section 5 presents three sets of equations, including the Euler equations, which can be solved using the UNIVERSE-series code under consideration. Section 6 presents various exact and approximate "Riemann Solvers." Section 7 presents detailed descriptions of the geometry treatment for three types of conservation cells (hexahedron, triangular prism, and tetrahedron). Section 8 deals with treatment of "topology" — the way the cells are connected together and how the connectivities change when new nodes and cells are introduced or links removed. Section 9 presents in detail the primary grid generation method (TRIM3D) that is used to generate 3-d unstructured tetrahedral meshes used in this effort as well as a briefer description of a 2-d unstructured grid method (TRIM2D). This section also includes a discussion of methods that can be employed to "smooth" the meshes as well as a consideration of issues involved in constructing meshes with higher-order

geometry information. Store-tracking and corresponding grid-adaptation issues are covered in Section 10. The links between boundary condition and geometry specifications as well as the specific implementation of boundary condition procedures in UNIVERSE-series codes at this time are presented in Section 11. Software architecture issues are discussed in Section 12. This includes reasons why the C language is particularly well suited to this effort. The various options and features possible under UNIVERSE-series algorithm and code framework are presented in Section 13. Several illustrative examples are presented in Section 14 including examples with scalar equations and the Euler equations, and problems in one, two and three dimensions. Section 15 includes a compilation of all cited references.

2.0 SUMMARY

During Phase I, the focus was on inviscid flows (2-d dynamic and 3-d static). A summary of progress for Phase I follows:

1. Three versions of a UNIVERSE-series flow solver were delivered to NWC. The first is UNIV1 and is written entirely in Fortran. The second is UNIVCF and includes sections in C which can be linked with the Fortran routines in UNIV1. The third is UNIVC which is almost totally written in the C language. It is expected that UNIVC will be the version to grow into Phase II. The exercise of developing the software in C has verified the expectation that the C language offers greater flexibility for implementing many needed versatile features.

2. A 2-d unstructured grid generation software (TRIM2D) has been delivered to NWC.

3. A 3-d unstructured grid generation software (TRIM3D) has been delivered to NWC.

4. The UNIVC flow solver includes a built-in 2-d plotting capability that can display results on Tektronix terminals. A pilot version of 3-d visualization software that is compatible with the flow solver and that runs on Silicon Graphics workstations has also been delivered to NWC.

5. A 3-d static inviscid CFD flow field computation around an F-18 configuration with pylon, rack and bombs has been demonstrated.

6. The inviscid flowfield for a 3-d missile in a 3-d cavity has been completed.

7. A store-tracking approach that exploits local grid adaptation (addition of cells, removal of short links, etc.) has been developed.

8. A 2-d case of a weapon in a bomb bay has been computed as a static case and the corresponding dynamic computation is in progress.

9. Several 1-d, 2-d and 3-d unit problems have been solved and are included in Section 14 as illustrative examples along with results for items 5–8 above.

10. Version description documents and user manuals have been written for UNIVC, TRIM3D, and TRIM2D. Similar information for the graphics software will be provided in Phase II when that software development has stabilized.

3.0 CONCLUSIONS

Section II will show that a new CFD framework has been developed which is more accurate and versatile than the previous state of the art for solving fluid dynamics problems involving complex geometries and complex flow phenomena such as shock waves. Some of the features of the numerical algorithms are listed here.

1. The grid generation method is able to automatically generate tetrahedral meshes for very complex topologies and the flow solver is able to accurately compute the solution on the resulting unstructured mesh. Three 3-d examples are provided in Section 14 (Space Shuttle multibody configuration, F-18 configuration with pylon and external stores, missile in a cavity).

2. Various levels of solution accuracy can be selected. Up to sixth-order spatial accuracy is available for 1-d problems and up to fourth-order spatial accuracy can be specified for 2-d and 3-d problems. Higher orders of solution accuracy can be implemented within the general framework if desired. Only second-order accuracy was required in Phase I but higher-order accuracy was folded in from the beginning anyway.

3. The solution accuracy must be combined with corresponding accuracy of geometry treatment. The geometry formulation is also extremely general and allows the specification and use of higher order shape functions to represent the geometry of each conservation cell.

4. The grid generation methodology must be able to provide such higher order mesh representation. During Phase I, standard linear tetrahedra were generated. In Phase II, a method to construct higher order geometry representations from such meshes will be implemented.

5. Several variations of Essentially nonoscillatory (ENO) interpolation techniques have been explored and implemented in order to evaluate the pros and cons of various approaches.

6. Several Riemann Solvers are included to permit the investigation of the significant properties of each. The exact Riemann Solver is computationally more expensive

than simpler ones. In the future, a strategy of automatically selecting the most inexpensive but adequate Riemann Solver for each cell or face based on the local flow behaviour can easily be considered because of the availability of such a spectrum of Riemann Solvers.

7. The UNIVC flow solver can solve 1-d, 2-d, axisymmetric and 3-d problems. This will enable a typical user to achieve a higher degree of proficiency in using the software because all problems can be solved using the same code. This also permits much more extensive validation of the methodology than is possible with separate codes.

8. The geometry specification is always assumed to be three dimensional. In order to perform a 1-d computation, a collection of 3-d cells strung along one dimension can be used. In addition to this, masks can be specified in order to avoid computing fluxes for all faces for a problem which is not 3-d. Proper symmetries can be specified for certain cell faces for problems with known symmetries (such as axisymmetric flow). The polynomials of the solution variables need only match the dimensionality requirement for the solution. For 1-d problems, for example, polynomials using powers of x can be selected; for 2-d problems polynomials in x and y are used; only for 3-d problems are all independent variables significant for the solution approximation. All these features result in the computational efficiency matching the complexity of the problem. Therefore, even though one code can do many types of problems, the computational time required is about as small for simpler (1-d, 2-d, or axisymmetric, for example) problems as codes constructed only for the simpler case.

9. The UNIVC flow solver can solve any desired set of equations. In particular, three equation sets are built in: a) the linear wave equation, 2) the nonlinear wave equation (inviscid Burgers' equation in one or more dimensions), and 3) the Euler equations of inviscid flow. This feature results in an ability to check out the properties of the numerical algorithm in controlled conditions and for a variety of physical phenomena in many disciplines, with each successful computation serving as corroboration for the other. For example, model equations such as the linear equation and Burgers' equation serve to test the properties of the method by direct comparison with known exact solutions. The ability to perform 1-d,

2-d and 3-d problems using the same code enhances this aspect.

10. A simple 2-d plotting capability is built into the flow solver for quick diagnosis of the solution. Eventually, this will be expanded to include 3-d plotting on graphics work stations. The goal of integrated post-processing capability will allow maximum consistency between numerical algorithm and visualization, analysis of results, etc.

11. The UNIVC code is constructed in such a fashion that memory required can be traded versus speed. For example, least-squares polynomial coefficients can be computed and stored at the beginning of the run (assuming mesh point locations and topology do not change). This increases the amount of required main memory but decreases the computational time. Similarly cell-face normals can either be computed and stored once for later repetitive use or can be computed upon demand.

12. It is therefore clear that many features have been built-in with both efficiciency and flexibility in mind. At this point the topological requirements for adapting the grid to motion of the external store is still in a state of flux and until the picture is clearer, the focus will be on modularity and flexibility rather than efficiency. However, computatational speed issues such as vectorization and parallelization on shared memory multiprocessors have been and are being consciously considered during all stages of code development.

13. A very flexible definition of "neighborhood" (which cells are near a given conservation cell) is part of the UNIVERSE-series code framework. In the specific implementation in UNIVC, only certain aspects of this flexibility are being exploited but the generality is anticipated to come in useful in future Phases.

14. A hierarchy of cell-face quadrature for integration of geometry and solution variables over cell boundaries is provided. For example, the mid-point quadrature formula for solution variables is useful for accuracies up to second order. For higher order accuracies, if the cell faces are planar, one can avoid computing more than one set of values for the cell-face normals even while using four-point quadrature formulae. In the most general and expensive situation, cell-face normals must be evaluated at each quadrature point. Once again, the focus is on

combining efficiency and generality in an optimal fashion.

15. The UNIVC flow solver can use three types of cells: hexahedral, triangular prism, and tetrahedral. This level of unification is unique. This feature allows the use of even existing structured grids while, at the same time, exploiting the improvements in solution accuracy possible within the new framework which employs true multi-dimensional interpolation even in second-order accuarcy mode. In this fashion, all the investment made over many years by most CFD organizations in developing or otherwise acquiring structured grid generation capability is exploited rather than wasted.

16. The framework includes user-selectable accuracy of temporal evolution. This can be suitably coupled with the spatial accuracy selection in such a fashion as to once again optimize the computational effort. For example, second-order spatial accuracy can be couple to first-order time accuracy for steady state problems and with second-order time accuracy for time-dependent problems. Fourth-order spatial accuracy may require the use of fourth-order Runge-Kutta time evolution operator for stability.

17. In the above, it is clear that the numerical framework and the implementation of the various software pieces have been selected to permit maximum flexibility and versatility. For example, higher order accuracy capability in the geometry and solution variables have been built in from the beginning even if during a given Phase or for a particular series of computations, all the features of the methodology will not necessarily be required.

SECTION II
TECHNICAL DISCUSSION

4.0 NUMERICAL FRAMEWORK FOR UNIVERSE-SERIES CODES

We now present background information about the new numerical capability that has been developed for the conservation laws of fluid dynamics. The approach is based on algorithms that can be of higher than second order accuracy which are embedded in a very flexible geometric framework that includes both structured and unstructured grids. While the cell shapes can be of any type, the book-keeping strategy is of the "unstructured" type and is even more versatile than popular unstructured-grid approaches. Codes developed using the new formulations will be part of the UNIVERSE-series. UNIVERSE denotes a Unification of essentially Nonoscillatory Interpolation techniques with a geometrically VERSatile implementation. The ENO (Essentially NonOscillatory)[1-4] interpolation facilitates the construction of numerical algorithms which can "capture" discontinuities such as shock waves and which are of arbitrarily high orders of accuracy, thereby transcending the inherent accuracy limitations of TVD schemes. Until recently ENO ideas were limited to either one-dimensional or Cartesian multi-dimensional applications or assumed smooth coordinate transformations.[5] UNIVERSE codes implement a truly multidimensional unrestricted version of ENO schemes that work with arbitrary cell shapes. For example, hexahedral, triangular prism and tetrahedral elements (conservation cells) can all be covered in a unified manner. This also implies that both structured and unstructured book-keeping schemes can be employed to conveniently treat complex topologies. New, fully automatic, unstructured grid generation methods have also been developed for integration with UNIVERSE-series flow solvers. Related material can also be found in Ref. 6.

Integral Form of Conservation Laws

The new algorithms have been developed for the general hyperbolic system of conservation laws represented by

$$\frac{\partial q}{\partial t} + \frac{\partial f_1}{\partial x} + \frac{\partial f_2}{\partial y} + \frac{\partial f_3}{\partial z} = 0 \tag{4.1}$$

which is in the "conservation-law form." The dependent (conserved) variables are denoted by q. The Cartesian coordinate directions (independent variables) are x,

y and z. The components of flux in the three coordinate directions are f_1, f_2 and f_3. An analogous formulation has also been developed for the quasi-linear or "non-conservation-law" form of the equations given by

$$\frac{\partial q}{\partial t} + A_1 \frac{\partial q}{\partial x} + A_2 \frac{\partial q}{\partial y} + A_3 \frac{\partial q}{\partial z} = 0 \quad . \tag{4.2}$$

However, in the present discussion, only the conservation-law form will be considered. Because of its applicability to "shock capturing", the conservation-law form is more general and appropriate numerical formulations based on it will be applicable to all flow-field regimes including subsonic and supersonic flows.

The conservation-law form shown in Eq. 4.1 is in the differential form. We now present the integral form of the conservation laws which can easily be derived from the differential form by integrating Eq. 4.1 with respect to x, y, z over any conservation cell whose volume is V.

$$\int\int\int_V \left(\frac{\partial q}{\partial t} + \frac{\partial f_1}{\partial x} + \frac{\partial f_2}{\partial y} + \frac{\partial f_3}{\partial z}\right) dx\, dy\, dz = 0 \quad . \tag{4.3}$$

This can be rewritten in vector notation as

$$\frac{\partial}{\partial t}\int\int\int_V q\, dx\, dy\, dz + \int\int\int_V (\vec{\nabla}\cdot\vec{F})\, dx\, dy\, dz = 0 \quad . \tag{4.4}$$

In the above,

$$\vec{F} = f_1\hat{j} + f_2\hat{k} + f_3\hat{l} \quad . \tag{4.5}$$

Applying the Gauss divergence theorem, we can convert the volume integral into a surface integral.

$$\frac{\partial}{\partial t}(\bar{q}V) + \int\int_S (\vec{F}\cdot\hat{n})\, dS = 0 \quad . \tag{4.6}$$

In the above equation, the cell average of the dependent variables are denoted by $\bar{q}$. The outward unit normal at any point of the boundary surface of a cell has been denoted by $\hat{n} = \hat{n}_x\hat{j} + \hat{n}_y\hat{k} + \hat{n}_z\hat{l}$.

$$\bar{q} = \frac{\int\int\int_V q\, dV}{\int\int\int_V dV} \quad . \tag{4.7}$$

The integral form of the conservation laws given by Eq. 4.6 defines a system of equations for the cell average values of the dependent variables. In order to construct numerical methods to solve the integral form of the conservation laws, we must be able to define cell geometries, approximate the dependent variables, develop spatial discretization procedures, develop time integration procedures to update the cell averages, etc. These topics are covered below.

Cell Geometry

A very general and flexible numerical formulation has been developed to solve the integral form of the governing conservation laws. While any cell shape definition can be used, details have been developed for three types in particular: (1) tetrahedral element, (2) triangular prism element, and (3) hexahedral element (Fig. 4.1). The name element will be used interchangeably with the term "conservation cell." The entire computational region is assumed to be divided up into a finite number of elements or conservation cells. The integral form of the conservation laws will be applied to each such cell. The hexahedral cell provides consistency with conventional structured-grid formulations. The triangular prism can be useful in situations where the grid generation can be generated plane by plane. The tetrahedral element will result in an inherently three-dimensional formulation. Encompassing these three types provides maximum flexibility in our approach to treat complex geometries.

Each element utilizes a local coordinate system (ξ, η, σ) (Fig. 4.1). Inside each element, the physical coordinates (x, y, z) are expressed as polynomials in terms of the local coordinate system. We now define three-dimensional, two-dimensional, and one-dimensional polynomials.

$$\begin{aligned} P_3(\cdot,\cdot,\cdot) &= \text{3-d polynomial} \\ P_2(\cdot,\cdot) &= \text{2-d polynomial} \\ P_1(\cdot) &= \text{1-d polynomial} \end{aligned} \tag{4.8}$$

A "shape function" approach is very useful in the definition of the cell geometry. The coordinates x, y, z are expanded in terms of the appropriate shape functions for

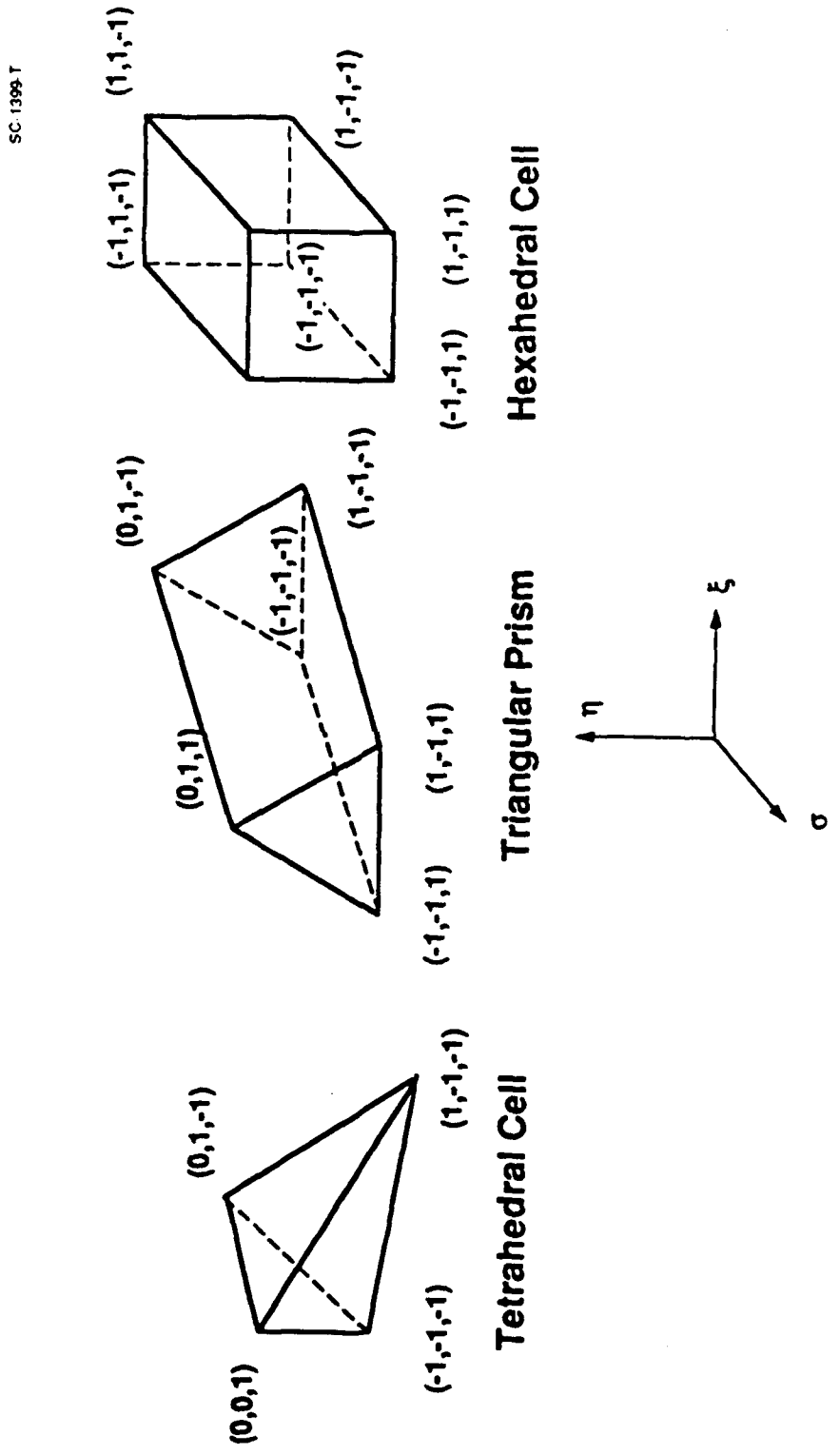


Fig. 4.1 Three types of conservation cells and their local coordinate system

each type of element. The geometry polynomials are defined for each cell and their range is restricted to that cell. Discontinuities in geometry can exist across each cell. Within each cell, by definition, the geometry will be smooth.

The simplest tetrahedral element comprises 4 nodes and 4 triangular faces. The geometry variables are expanded in terms of a single three-dimensional polynomial in the local variables.

$$\begin{aligned} f(\xi,\eta,\sigma) &= P_3(\xi,\eta,\sigma) \\ &= \sum_{\text{nodes}} N_i(\xi,\eta,\sigma)\, f_i \end{aligned} \tag{4.9}$$

The simplest triangular prism element comprises 6 nodes, 2 triangular faces and 3 quadrilateral faces (a total of 5 faces). The geometry variables are expanded in terms of the tensor product of a two-dimensional polynomial and a one-dimensional polynomial.

$$\begin{aligned} f(\xi,\eta,\sigma) &= P_2(\xi,\eta)P_1(\sigma) \\ &= \sum_{\text{nodes}} N_i(\xi,\eta,\sigma)\, f_i \end{aligned} \tag{4.10}$$

The simplest hexahedral element comprises 8 nodes and 6 quadrilateral faces. The geometry variables are defined in terms of the tensor product of three one-dimensional polynomials.

$$\begin{aligned} f(\xi,\eta,\sigma) &= P_1(\xi)P_1(\eta)P_1(\sigma) \\ &= \sum_{\text{nodes}} N_i(\xi,\eta,\sigma)\, f_i \end{aligned} \tag{4.11}$$

Higher order elements (Fig. 4.2) can also be defined and used but will not be considered in this discussion or in the proposed effort. In the above, f can be x, or y or z. Each of the polynomials on the right hand side of Eqs. 4.9, 4.10 and 4.11 can be of arbitrary degree.

Given nodal values of x, y, z, we can evaluate the geometry polynomials at any point of the cell that has been defined in terms of the local coordinate system. In particular, we will need to find (x, y, z) coordinates of quadrature points on each face

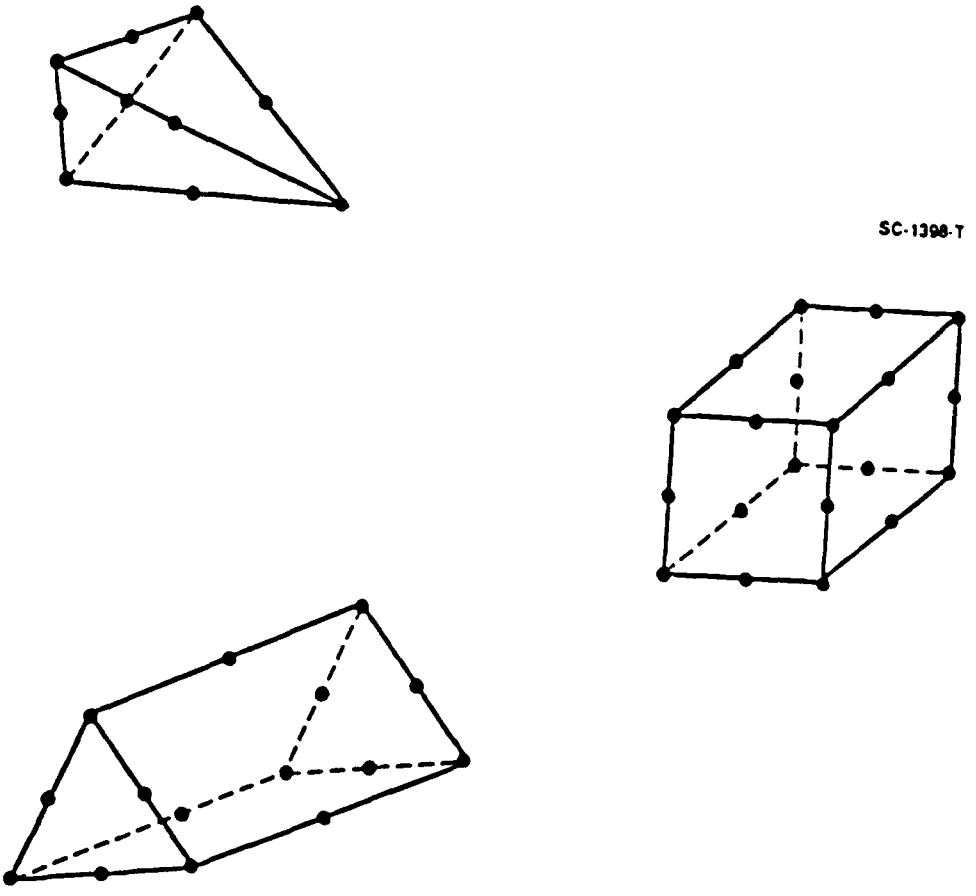


Fig. 4.2 Examples of higher order elements

as well as the components of the outward point normal at these quadrature points. The geometry polynomials also yield the metrics x_ξ, y_ξ, z_ξ, x_η, y_η, z_η, $x_\sigma, y_\sigma, z_\sigma$. In order to compute the outward pointing normal at any point of a given face of a given cell, the following steps are defined.

1) Obtain two linearly independent vectors on the face passing through the given point. Such vectors can easily be defined in terms of the local metrics given above.

2) The cross product of these two vectors is in the direction of the normal to the face.

Finite-Element-Like Formulation

The basic numerical framework is based on a finite-element-like (FEL) formulation.

a) We have already seen that the conservation cells are finite elements.

b) In this section, we will approximate the variation of dependent variables within each cell using polynomials of the physical independent variables x, y, z.

However, the CFD formulation to be presented here differs in several ways from traditional finite-element (FE) approaches.

c) The CFD FEL approach will use piecewise polynomial interpolation for the dependent variables. These polynomials can be discontinuous across cell boundaries; for smooth solution behavior, the difference between left and right values at a cell face common to two cells will be small. Such a piecewise polynomial approach facilitates "capturing" discontinuities.

d) The discretization of the governing equations follows directly from the integral form of the conservation laws. No variational principle or method of weighted residuals or other indirect approach is employed.

e) The dependent variable polynomials are directly written in terms of x, y, z and not in terms of the element coordinate system variables ξ, η, σ. The domain and

range of the geometry polynomials is restricted to a single cell. The domain of the dependent variable polynomials is not thus restricted. We will soon see that for the purpose of evaluating the polynomial coefficients, the dependent variable polynomials are evaluated over many neighboring cells. At cell faces common to two or more cells, the overlapping polynomial definitions lead to the piecewise polynomial approach and helps define left and right states before resolving them using a Riemann solver.

To provide capability to compute 1-d, 2-d and 3-d flows efficiently, the dependent variable polynomials are defined to be

$$f(x,y,z) = P_3(x,y,z) \tag{4.12a}$$

or

$$f(x,y) = P_2(x,y) \tag{4.12b}$$

or

$$f(x) = P_1(x) \tag{4.12c}.$$

The geometry is always defined in 3-d space. Using the appropriate dependent variable polynomial type, the solution can be forced to be 1-d, 2-d, or 3-d. In this manner, we can easily compute 1-d, 2-d, and 3-d solutions with the same flow solver. All that is required in 1-d, for example, is to string together a 1-d collection of cells. In the present effort, we do not consider tensor product polynomials for the dependent variables. However, these and other approaches have been developed and applied by Science Center CFD researchers elsewhere.

In what follows, we only consider the case of three-dimensional dependent variable polynomials for the sake of brevi.. of presentation. The two- and one-dimensional simplifications can be derived in straight-forward fashion.

We use a hierarchy of polynomials depending on the desired accuracy. The simplest involves only the constant term.

$$P_3(x,y,z) = p_0 \quad . \tag{4.13}$$

The next level includes linear terms.

$$P_3(x,y,z) = p_0 + p_1\,x + p_2\,y + p_3\,z \tag{4.14}$$

The next level includes quadratic terms also.

$$\begin{aligned} P_3(x,y,z) &= p_0 + p_1\,x + p_2\,y + p_3\,z + p_4\,x^2 + p_5\,y^2 + p_6\,z^2 + p_7\,xy + p_8\,yz + p_9\,zx \\ &= \sum_{i=0}^{I} p_i\,x^{j(i)}\,y^{k(i)}\,z^{l(i)} \end{aligned} \tag{4.15}$$

Within the general framework, even higher order polynomials can be utilized. However, it is anticipated that quadratic polynomials will result in accuracies greater than conventionally achieved in general purpose flow solvers applicable to complex geometry problems. In the UNIVC flow solver, cubic polynomials are also selectable in two and three dimensions and even fourth and fifth degree polynomials can be used in one dimensional problems. In Phase I, computations for large scale problems were limited to second-order accuracy. However, several unit problem examples were computed with higher-degree piecewise polynomials. Piecewise linear polynomials result in second-order accuracy, quadratic polynomials lead to third-order accuracy, and so on. In Phase II, piecewise quadratic and cubic polynomials will be explored in more depth to analyse the advantages/disadvantages of using increased order of accuracy.

In dealing with the integral form of the conservation laws, we must deal with cell averages of the dependent variables. If we express the dependent variables as polynomials, we must deal also with cell averages of the polynomials.

$$\begin{aligned} \frac{1}{V}\int\int\int_V P_3(x,y,z)\,dV &= \frac{1}{V}\left(\sum_{i=0}^{I} p_i \int\int\int_V x^{j(i)}\,y^{k(i)}\,z^{l(i)}\,dx\,dy\,dz\right) \\ &= \sum_i a_i\,p_i \end{aligned} \tag{4.16}$$

In the above we see that the average value of a polynomial taken over an element or conservation cell can be expressed as a weighted sum of the polynomial coefficients.

The weights a_i depend on the shape of the particular cell over which the average is defined.

$$a_i = \frac{\int\int\int_V x^{j(i)}\, y^{k(i)}\, z^{l(i)}\, dV}{\int\int\int_V dV} \tag{4.17}$$

It is convenient to define the weights in terms of the gradient of a vector $\vec{X}$ defined in turn by

$$\vec{X}_i = \frac{1}{3}\left[\frac{x^{j(i)+1}}{j(i)+1} y^{k(i)} z^{l(i)}\,\hat{j} + x^{j(i)}\frac{y^{k(i)+1}}{k(i)+1} z^{l(i)}\,\hat{k} + x^{j(i)} y^{k(i)}\frac{z^{l(i)+1}}{l(i)+1}\,\hat{l}\right] \tag{4.18}$$

In terms of this vector, the weights a_i are given by

$$\begin{aligned} a_i &= \frac{1}{V}\int\int\int_V (\vec{\nabla}\cdot\vec{X}_i)\, dV \\ &= \frac{1}{V}\int\int_S (\vec{X}_i\cdot\hat{n})\, dS \end{aligned} \tag{4.19}$$

In the above Eq. 4.19, we have been able to simplify the volume integration to a surface integration in the usual way by applying the Gauss divergence theorem. Comparing Eq. 4.6 and Eq. 4.19, we see that the surface integration formula applies to both the dependent variable flux vector $\vec{F}$ and the "geometry flux vector" $\vec{X}$.

In what follows, numerical procedures developed to approximate surface integrals must deem to apply to either type of vector. We first divide the surface integral into component parts that apply over each distinct face or side of any cell under consideration. We then replace the integral on each face with a numerical quadrature.

$$\begin{aligned} \int\int_S (\vec{F}\cdot\hat{n})\, dS &= \sum_{\text{faces}} \int\int_F (\vec{F}\cdot\hat{n})\, dS \\ &= \sum_{\text{faces}} \sum_{\text{quads.}} \vec{F}_i\cdot\hat{n}_i\, S_i \end{aligned} \tag{4.20}$$

Here, "quads." is an abbreviation for "quadrature points." For second-order applications, we use the midpoint rule for quadrature for the solution flux integration and a 4-point Gaussian quadrature for computing the geometry weights a_i. For higher-order calculations, the higher-order accurate quadrature formulae are used for the solution

flux integration. The weights of the quadrature formulae must include the effect of cell surface area corresponding to the given face and such weights are denoted by S_i in the above equation. The midpoint formula can be represented as

$$\int\int_S (\vec{F} \cdot \hat{n})\, dS = \sum_{\text{faces}} \vec{F}_m \cdot \hat{n}_m\, S_m \tag{4.21}$$

where m denotes the centroid of each face. Even higher-order quadrature formulae may be employed if necessary in the future.

Outline of Solution Procedure

The original initial value problem (IVP) for the differential form of the conservation laws specifies initial values of the dependent variables. In the IVP for the integral form of the conservation laws, initial values of cell averages of the dependent variables will be defined. Given such initial values, the three steps used to set up the discretization procedure for the integral form of the conservation laws are as follows.

(i) Define dependent variable polynomials in each cell so that the cell average of the polynomial approximation matches the cell average of the dependent variable which is either given as part of the initial value specification or obtained by updating the cell averages during subsequent steps of the solution process. This process of defining pointwise polynomial behavior from known values of cell averages is called the "reconstruction procedure" and is the major subject of the next section.

(ii) Evaluate the polynomials at all quadrature points. This will lead to "left" and "right" values at each quadrature point which lies on a face common to two cells.

(iii) Construct the solution of a local Riemann problem using these left and right states and from this evaluate the numerical flux. Use such numerical fluxes in Eq. 4.20 or Eq. 4.21 and evaluate the right hand side of

$$\frac{\partial}{\partial t}(\bar{q}V) = -\int\int_S (\vec{F} \cdot \hat{n})\, dS \tag{4.22}$$

(iv) Steps (i)–(iii) complete the discretization of the right hand side of Eq. 4.22. To the resulting semi-discrete system of equations, we apply a suitable time-integration procedure such as Heun's method (for second-order accuracy) or the fourth-order accurate Runge-Kutta scheme.

Reconstruction of Piecewise Polynomials

For cell C, the corresponding polynomial may be written as

$$P_3^C(x,y,z) = \sum_{i=0}^{np} p_i^C x^{j(i)} y^{k(i)} z^{l(i)} \tag{4.23}$$

In order to define this polynomial, we need to know the polynomial coefficients p_i^C which must be determined by a suitable procedure. One of the conditions that this polynomial must satisfy is

$$\overline{P}_3^C = \sum_i a_i^C p_i^C = \bar{q}_C \tag{4.24}$$

In addition to Eq. 4.24, we need np more appropriate equations to solve for $p_i^C, i = 0,\ldots,np$. For example, if we match the cell average of $P^C(x,y,z)$ with $\bar{q}_N$ over np neighboring cells as well as for cell C, we will obtain a total of $np+1$ equations which will be sufficient to solve for all $(np+1)$ coefficients of the polynomial.

$$\sum_{i=0}^{np} a_i^n p_i^C = \bar{q}_N \quad N = C, N_1, N_2, \ldots, N_{np} \tag{4.25}$$

This leads to the matrix system of equations given by

$$AP = Q \tag{4.26}$$

where A is matrix of size $(np+1) \times (np+1)$, P is the set of unknown polynomial coefficients, and Q is the collection of known cell averages of the dependent variables at $(np+1)$ cells including and in the neighborhood of cell C.

Neighbors

The UNIVERSE-series CFD formulation defines a "neighbor" of a given cell in a very flexible and useful way.

First, we consider two types of cell connectivities (Fig. 4.3):

(1) Node-aligned cells (NAC)

(2) Surface-aligned cells (SAC)

Next, we consider different types of neighbors:

(1) Touching neighbors (TN)

These include

(1a) Common-node neighbor (CNN)

(1b) Common-face neighbor (CFN)

(1c) Touching-face neighbor (TFN)

(2) Proximity neighbors (PN)

This latter type is defined in terms of distance from a given cell.

Neighborhood Hierarchy

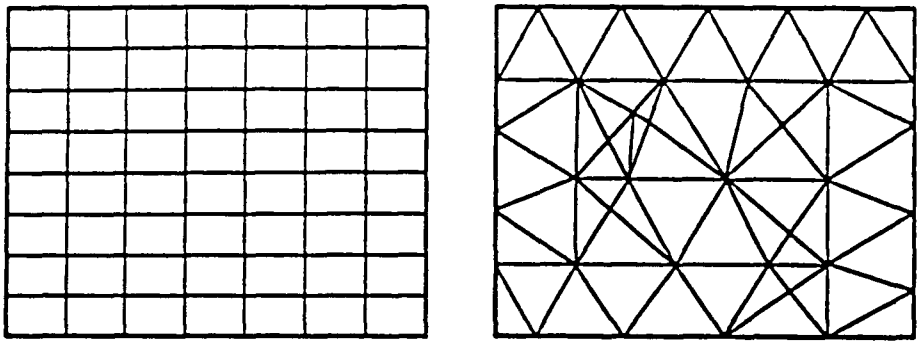
A neighborhood is now defined to be a collection of neighboring cells.

H^0 is the cell itself.

H^1 is the cell and its neighbors.

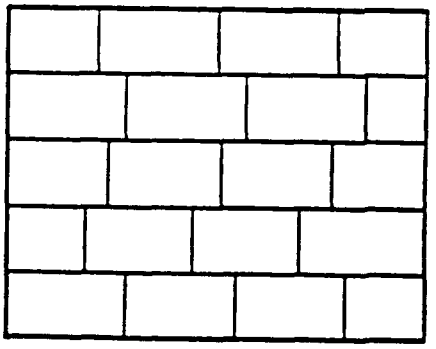
H^2 is defined as the union of H^1 and the neighbors of all the cells in H^1.

This process may be continued recursively.



Node Aligned Cells
(NAC)

SC-1396-T



Surface Aligned Cells
(SAC)

Fig. 4.3 Node-aligned and surface aligned cells

Least-Squares Reconstruction

Let us now consider a neighborhood of cells constructed hierarchically from one central cell. It is, in general, true that any neighborhood hierarchy may either contain less than or greater than $(np + 1)$ cells and not just exactly $(np + 1)$ cells. We have already considered the case when we have exactly the number of cells required to solve for the number of polynomial coefficients. Now, let us consider the case when we have a neighborhood with $(nc + 1)$ cells where $nc > np$. The previous hierarchical neighborhood is assumed to have less than $(np + 1)$ cells. In this case, we have an overdetermined set of equations $AP = Q$ where A is an $(nc + 1) \times (np + 1)$ matrix and Q is a set of $(nc + 1)$ cell averages.

We can solve this set of equations in the context of a generalized inverse using the least squares method. We first subtract the equation for cell C from all the remaining equations. Let the resulting set of equations (with the row and column corresponding to cell C removed from consideration temporarily) be

$$\overline{A}\,\overline{P} = \overline{Q} \tag{4.27}$$

It is clear that $\overline{A}$ is a matrix of size $nc \times np$. In the least-squares method, we now solve the generalized system of equations

$$\overline{A}^T\overline{A}\,\overline{P} = \overline{A}^T\overline{Q} \tag{4.28}$$

By solving Eq. 4.28, we obtain np polynomial coefficients. Substituting these into the equation for cell C we have separated out earlier, we obtain the remaining polynomial coefficients. The matrix $\overline{A}^T\overline{A}$ is symmetric.

The resulting reconstructed polynomial has the property that its average matches the cell average $\bar{q}_C$ but it does not necessarily match any other cell average. However, the average of the polynomial taken over the region defined by each cell in the neighborhood is a good fit to all the corresponding cell averages in the "least-squares" sense. This procedure yields an eminently satisfactory algorithm for smooth flows. For another reference for least-squares quadratic reconstruction, refer to Barth.[7]

ENO Reconstruction

Once again, let us consider an overdetermined set of equations to solve for the polynomial coefficients. Now, we seek to obtain a "best" polynomial rather than a "least-squares" one. The polynomial should result in an Essentially NonOscillatory (ENO) interpolation. As always, the equation for cell C must be satisfied. From the remaining nc equations, we can select any combination of np equations and solve the resulting set of $np + 1$ equations. There are

$$\binom{nc}{np}$$

such combinations. The combination that yields the best one in terms of its ENO property is to be preferred. For example, when the flow field contains a single shock wave, the neighbors selected should lie on the same side of the shock as cell C. This approach may be termed the "best stencil" formulation and has been applied very successfully in various forms to structured grid ENO formulations. Reference 8 contains many different approaches to this task.

Alternatively, a "best term" approach has also been tried out. In this formulation, the least-squares polynomials are first selected for all cells. Each coefficient of the polynomial in a given cell corresponds to the appropriate derivative of the polynomial (up to a constant coefficient) evaluated at the centroid of the cell. We replace each term obtained by using the least-squares formulation by the same term (derivative up to a constant) evaluated at the center of the cell but computed using a neighboring cell's polynomial if certain conditions are met: e.g., the absolute value of the term due to the neighbor polynomial times a factor greater than one is less than the absolute value of the term due to the polynomial in the central cell. This procedure is performed in such a way as to give preference to the original least-squares polynomial.

For one-dimensional shock-tube problems, it has often been demonstrated that it is better to select the best stencils based on comparing interpolates of local characteristic variables and not the conserved dependent variables. However, within the context of unstructured grid formulations this approach is very expensive and consideration of such issues is postponed for future work.

Polynomial Evaluation at Quadrature Points

Recall that quadrature points on the surface of a cell are located at the mid points of the faces.

(a) For CFN type of neighboring cells, at a common face the quadrature points from either cell are at the same physical location (Fig. 4.4) .

(b) For TFN type of neighboring cells, on a touching face the corresponding quadrature points may not coincide (Fig. 4.4). "Left" and "right" values must be computed at each of the quadrature points and used in the flux integration. A more sophisticated approach can also be used to identify common parts of touching faces and the boundary quadratures rewritten as a sum of quadratures over such common regions. TFN type of cells have not been implemented in the UNIVC code and further consideration of related ideas is avoided in this report.

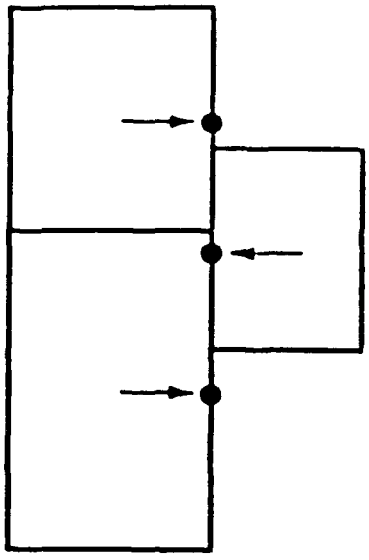
Riemann Solver

At every quadrature point, we have a local Riemann problem. We use any appropriate exact or approximate Riemann solver to resolve the left and right states and construct a numerical flux. For example, Roe's,[9-13] Godunov's,[13-14] Osher's[13,15-17] or even a Lax-Friedrichs or Rusanov type Riemann solver can be employed and in fact have been incorporated into the UNIVERSE-series flow solvers provided during Phase I to NWC. NWC's Dr. Burman has also added another Riemann solver due to Harten and Lax[18] and this has also been added to the UNIVC code. The importance of the Riemann solver decreases with increasing degree of interpolation for smooth problems.

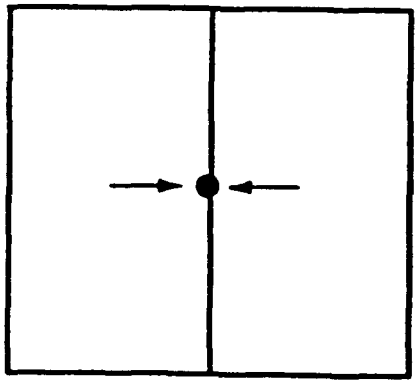
Time Stepping Scheme

In this paper, we have taken a semidiscrete approach. Other alternatives are given in Refs. 8 and 19. A second-order time-accurate formulation is given below as an example. This is fashioned after Heun's method or the second-order Runge-Kutta

SC-1397-T



Surface Aligned Cells



Node Aligned Cells

Fig. 4.4 Quadrature points for two types of neighbors

method (RK2). The RK4 method can be implemented in similar fashion. Higher than second-order spatial accuracy results in reduced numerical dissipation and this sometimes necessitates the use of the fourth-order Runge-Kutta formulation which has a larger stability range than the second-order Runge-Kutta method.

In semidiscrete form, the equations to be solved are

$$\frac{\partial}{\partial t}(\bar{q}V) = RHS(\bar{q}, t) \tag{4.29}$$

The corresponding time stepping method can be written as

$$\begin{aligned} (\bar{q}V)^1 &= (\bar{q}V)^n + RHS(\bar{q}^n, t^n)\Delta t \\ (\bar{q}V)^{n+1} &= \frac{1}{2}\left[(\bar{q}V)^n + (\bar{q}V)^1 + \Delta t\, RHS(\bar{q}^1, t^{n+1})\right] \end{aligned} \tag{4.30}$$

The fourth-order accurate Runge-Kutta scheme can be written as

$$\begin{aligned} k_1 &= RHS(\bar{q}^n, t^n) \\ (\bar{q}V)^1 &= (\bar{q}V)^n + \frac{\Delta t}{2}k_1 \\ k_2 &= RHS(\bar{q}^1, t^n + \frac{\Delta t}{2}) \\ (\bar{q}V)^2 &= (\bar{q}V)^n + \frac{\Delta t}{2}k_2 \\ k_3 &= RHS(\bar{q}^2, t^n + \frac{\Delta t}{2}) \\ (\bar{q}V)^3 &= (\bar{q}V)^n + \Delta t\, k_3 \\ k_4 &= RHS(\bar{q}^3, t^{n+1}) \\ (\bar{q}V)^{n+1} &= (\bar{q}V)^n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned} \tag{4.31}$$

In the above, the explicit dependence of RHS on t is useful for time-dependent problems where the boundary conditions or other behavior explicitly depends on time.

5.0 CONSTITUTIVE EQUATIONS

The Conservation Laws

The principles of conservation of mass, momentum, and energy are fundamental in the study of fluid dynamics and similar problems. Model conservation laws that are simpler than the Euler or Navier-Stokes equations of fluid dynamics can be useful to study certain aspects of the whole problem. In this section we present a linear model equation, a nonlinear model equation and the Euler equations (for perfect gas). A conservation law represents the principle that the conserved quantity increases or decreases depending on whether there is a net positive or negative flux of the quantity into the control volume being considered. When the size of cubical control volume is permitted to shrink to zero along each one of the Cartesian coordinate directions, if continuity of the dependent variables is assumed, the differential conservation law form of the equations shown as Eq. 4.1 results. The integral form of the conservation laws shown in Eq. 4.6 truly represents the original statement of the conservation principle and in that sense can be thought of as the basic form from which the differential form can be derived by considering the limit case of cell volume tending to zero. However, since one is able to go from one form to the other using the steps outlined at the beginning of Section 4 and their reverse, the particular form of the equations used is of secondary importance. However, the differential form of the equations serve as a convenient means to catalog the constitutive equations being discussed and that is the approach taken here.

The Strong-Conservation-Law Form

The conservation law form of the equations (Eq. 4.1), is known as the strong-conservation-law form. When continuity of the dependent variables can be assumed, the equations can be rewritten in the "non-conservation-law" form given by Eq. 4.2. The strong conservation law form of the equations is closely connected with the existence of "weak" solutions,[20] i.e. composite solutions composed of continuously differentiable parts which satisfy the differential form (either the strong-conservation-law form or the non-conservation-law form is sufficient for this purpose) together with jump discontinuities that satisfy the appropriate jump conditions. In fact the

jump conditions are derived from the strong-conservation-law form. Since more than one strong-conservation-law form of the equations — all of which are equivalent for differentiable functions — may be derivable, the specific form of the conservation laws (in particular, the choice of the dependent variables) is significant. The choice that corresponds most directly with the physical processes of mass, momentum, and energy conservation are, in that sense, the most suitable.

Numerical methods are constructed for the strong-conservation-law form of the equations in such a manner that they obey a "discrete conservation principle" (single numerical flux per cell face) that can be exploited to "capture" shock waves and other discontinuities. The numerical framework described in the earlier section satisfies this requirement. Such numerical methods can compute the solution in smooth regions of the flowfield to the desired order of accuracy. Additionally, they can, without explicitly making use of the shock-jump relations, automatically compute across discontinuities. In order to achieve the required order of accuracy in smooth regions, polynomial interpolation using the appropriate degree of reconstruction is employed in the numerical framework outlined in Section 4. The piecewise polynomial approach is utilized to permit the approximation of solutions with discontinuities. In order to avoid undesirable "spurious" oscillations (oscillations that are not to be expected in the exact solution), the ENO scheme is employed at the approximation level. This is coupled to "Riemann Solvers" that represent the physics of hyperbolic conservation laws to the desired degree of fidelity. Taken together, these aspects of the numerical framework result in the ability to compute both smooth and weak solutions accurately and with high fidelity.

The Weak-Conservation-Law Form

Having introduced the term "strong-conservation-law" form of the equations, we now present a brief discussion of the so-called "weak-conservation-law" form for the sake of completeness and clarification. The weak-conservation-law form is derived

from Eq. 4.1 by applying the coordinate transformations

$$\begin{aligned} \tau &= t \\ \xi &= \xi(x,y,z,t) \\ \eta &= \eta(x,y,z,t) \\ \sigma &= \sigma(x,y,z,t) \end{aligned} \tag{5.1}$$

to obtain

$$\begin{aligned} &\frac{\partial q}{\partial \tau} + \xi_t \frac{\partial q}{\partial \xi} + \eta_t \frac{\partial q}{\partial \eta} + \sigma_t \frac{\partial q}{\partial \sigma} \\ &\quad + \xi_x \frac{\partial f_1}{\partial \xi} + \eta_x \frac{\partial f_1}{\partial \eta} + \sigma_x \frac{\partial f_1}{\partial \sigma} \\ &\quad + \xi_y \frac{\partial f_2}{\partial \xi} + \eta_y \frac{\partial f_2}{\partial \eta} + \sigma_y \frac{\partial f_2}{\partial \sigma} \\ &\quad + \xi_z \frac{\partial f_3}{\partial \xi} + \eta_z \frac{\partial f_3}{\partial \eta} + \sigma_z \frac{\partial f_3}{\partial \sigma} = 0 \end{aligned} \tag{5.2}$$

While in years past, this form of the equations has been subject of debate as to its use in constructing shock-capturing methods, we ignore its significance (if any) here because it is a) unnecessary (we do not use coordinate transformations for the conservation laws) and b) it does not directly lead to discrete conservation.

We now catalog the three sets of equations that can be selected for solution in UNIVERSE-series codes such as UNIVC. The corresponding Riemann Solvers are presented in the next section.

Linear Wave Equation

The multi-dimensional form of the linear wave equation for a scalar variable q fits Eq. 4.1 with

$$f_1 = a\,q, f_2 = b\,q, f_3 = c\,q \tag{5.3}$$

where a, b, and c are constants (usually $\equiv 1$).

Inviscid Burgers' Equation

The multi-dimensional form of the inviscid Burgers' equation for a scalar variable q fits Eq. 4.1 with

$$f_1 = a\frac{q^2}{2}, f_2 = b\frac{q^2}{2}, f_3 = c\frac{q^2}{2}, \tag{5.4}$$

where a, b, and c are constants (usually $\equiv 1$).

Euler Equations

The Navier-Stokes equations can be written in three-dimensional Cartesian coordinates using the conservation-law form notation as

$$\frac{\partial q}{\partial t} + \frac{\partial(F_1 - G_1)}{\partial x} + \frac{\partial(F_2 - G_2)}{\partial y} + \frac{\partial(F_3 - G_3)}{\partial z} = \Omega \tag{5.5}$$

where

$$q = \begin{pmatrix} e \\ \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho\sigma_1 \\ \vdots \\ \rho\sigma_N \end{pmatrix}, F_1 = \begin{pmatrix} (e+p)u \\ \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ \rho u\sigma_1 \\ \vdots \\ \rho u\sigma_N \end{pmatrix}, F_2 = \begin{pmatrix} (e+p)v \\ \rho v \\ \rho vu \\ \rho v^2 + p \\ \rho vw \\ \rho v\sigma_1 \\ \vdots \\ \rho v\sigma_N \end{pmatrix}, F_3 = \begin{pmatrix} (e+p)w \\ \rho w \\ \rho wu \\ \rho wv \\ \rho w^2 + p \\ \rho w\sigma_1 \\ \vdots \\ \rho w\sigma_N \end{pmatrix},$$

In comparing the above equation (Eq. 5.5) with Eq. 1, we see that $f_1 = (F_1 - G_1)$, etc. The Euler equations can be obtained from the above by setting $G_1 = G_2 = G_3 = 0$. The Euler equations form a hyperbolic set of conservation laws. For perfect gas, we can express the pressure p in terms of the dependent variables as

$$p = (\gamma - 1)(e - \frac{1}{2\rho}((\rho u)^2 + (\rho v)^2 + (\rho w)^2)) \tag{5.6}$$

The viscous terms are

$$G_1 = \begin{pmatrix} K\frac{\partial T}{\partial x} + u\tau_{xx} + v\tau_{xy} + w\tau_{xz} \\ 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xz} \\ \rho D\frac{\partial \sigma_1}{\partial x} \\ \vdots \\ \rho D\frac{\partial \sigma_N}{\partial x} \end{pmatrix} \tag{5.7a}$$

$$G_2 = \begin{pmatrix} K\frac{\partial T}{\partial y} + u\tau_{xy} + v\tau_{yy} + w\tau_{yz} \\ 0 \\ \tau_{xy} \\ \tau_{yy} \\ \tau_{yz} \\ \rho D\frac{\partial \sigma_1}{\partial y} \\ \vdots \\ \rho D\frac{\partial \sigma_N}{\partial y} \end{pmatrix} \tag{5.7b}$$

$$G_3 = \begin{pmatrix} K\frac{\partial T}{\partial z} + u\tau_{xz} + v\tau_{yz} + w\tau_{zz} \\ 0 \\ \tau_{xz} \\ \tau_{yz} \\ \tau_{zz} \\ \rho D\frac{\partial \sigma_1}{\partial z} \\ \vdots \\ \rho D\frac{\partial \sigma_N}{\partial z} \end{pmatrix} \tag{5.7c}$$

and

$$\begin{aligned} \tau_{xx} &= 2\mu\frac{\partial u}{\partial x} - \frac{2}{3}\mu\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) \\ \tau_{yy} &= 2\mu\frac{\partial v}{\partial y} - \frac{2}{3}\mu\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) \\ \tau_{zz} &= 2\mu\frac{\partial w}{\partial z} - \frac{2}{3}\mu\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}\right) \\ \tau_{xy} &= \mu\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) \\ \tau_{xz} &= \mu\left(\frac{\partial u}{\partial z} + \frac{\partial w}{\partial x}\right) \\ \tau_{yz} &= \mu\left(\frac{\partial w}{\partial y} + \frac{\partial v}{\partial z}\right) \end{aligned} \tag{5.7d}$$

The source term vector can be expanded to

$$\Omega = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \text{source term} \\ \vdots \\ \text{source term} \end{pmatrix} \tag{5.8}$$

The Navier-Stokes equations will be utilized in Phase II and is therefore presented here for convenience. The first five equations in Eq. 5.5 represent the usual conservation laws for total energy per unit volume (e), mass per unit volume (ρ), and the three components of momentum per unit volume ($\rho u, \rho v, \rho w$). In Phase II, when the Reynolds-averaged form of the Navier-Stokes equations will be used to simulate turbulent flows, the additional conservation equations will be used to represent turbulent kinetic energy, diffusivity, etc., as part of a two-equation turbulence model.

Other Equations

Since the UNIVERSE-series formulation is very flexible and the software implementations very modular, other sets of equations can easily be plugged in. Two examples are 1) the equations of acoustics and 2) Maxwell's equations of electromagnetics. The additional species equations along with the corresponding source terms can also be used to represent the chemically reacting species of a finite-rate-chemistry computation. Complex equations of state may be used to represent equilibrium flows. Eventually, therefore, UNIVERSE-series codes can replace the USA-series of codes.[21-35]

6.0 RIEMANN SOLVERS

General Framework

The Riemann problem is an initial value problem (IVP) with piecewise-constant initial data (one set to the left of the origin and the other to the right). It is similar to the "shock tube problem" of gasdynamics where at $t = 0$, a diaphragm separates the left state from the right. In the shock tube, the bursting of the diaphragm brings the left and right states into contact. In the mathematical statement of the Riemann Problem, we assume that the left and right states are separated at $t = 0$ and at $t > 0$, we let the two states interact. In the shock-tube problem, typically the initial velocities are both zero. In the Riemann problem statement, we have the freedom to specify arbitrary values for these. The Riemann problem can be constructed in a meaningful way for any set of hyperbolic conservation laws while the shock-tube problem is related directly only to the solution of the Euler equations.

"Riemann Solver" is the name given to the procedure that constructs the solution to the Riemann problem. In the one-dimensional case, knowing the solution implies a quantitative and qualitative knowlege of $q(x,t)$ for $-\infty < x < +\infty$ and $t > 0$. It turns out (for the piecewise-constant case under discussion) that the solution is self-similar in the variable $(\theta = x/t$. Therefore $q(x,t) = q^R(\theta)$, with the superscript R denoting the solution to the Riemann problem.

Jump Conditions

We now present some facts and perspectives regarding the solution of the Riemann problem in one spatial dimension corresponding to the one-dimensional system of conservation laws given by

$$\frac{\partial q}{\partial t} + \frac{\partial f(q)}{\partial x} = 0 \tag{6.1}$$

where q and f are m-vectors. If the left and right states are equal, we obtain the degenerate Riemann problem for which the solution is

$$q^R(\theta) = q_r = q_l \tag{6.2}$$

where the subscripts r and l refer to the right and left states respectively. In the nontrivial case of $q_r \neq q_l$, we begin (at $t = 0$) with discontinuities in one or more of the dependent variables. Discontinuities must satisfy the jump relations (see Whitham[20])

$$-\dot{x}\,[q] + [f(q)] = 0 \quad . \tag{6.3}$$

The notation $[\cdot]$ denotes the jump $(\cdot)_r - (\cdot)_l$ where "$\cdot$" refers to any quantity. The speed of propagation of the discontinuity is given by $\dot{x}$.

One-Dimensional Euler Equations

Let us consider an example case for the one-dimensional Euler equations.

$$\begin{aligned} -\dot{x}\,[e] + [(e+p)u] &= 0 \\ -\dot{x}\,[\rho] + [\rho u] &= 0 \\ -\dot{x}\,[\rho u] + [p + \rho u^2] &= 0 \end{aligned} \tag{6.4}$$

If the elements of q_l are known, the jump relations given above provide m (= 3 for the 1-d Euler equations) algebraic relations for $m+1$ unknowns (the m elements of q_r and $\dot{x}$). Therefore, we have a one-parameter family of solutions possible. For example, if $\dot{x}$ is chosen as the free parameter, the number of equations matches the remaining number of unknowns. We can also select one of the elements of q_r or a particular combination of them as the free parameter. For the Euler equations, therefore, we can select p_r or ρ_r or u_r as the free parameter; we then use the three jump relations to solve for $(\dot{x}, \rho_r, u_r)$ or $(\dot{x}, p_r, u_r)$ or $(\dot{x}, p_r, \rho_r)$, respectively. Even if the number of equations is sufficient (once one free parameter is chosen), the choice of the free parameter must lead to a solvable set of equations. For nonlinear equations, more than one solution may also be found. We also observe that in a similar fashion, the elements of q_l may be treated as unknowns and the elements of q_r as the known values. In fact, the sets of "knowns" and "unknowns" may also be a mixed set of left and right states as long as there are m known values, m unknown values and a free parameter and the resulting set of equations for the unknowns are solvable.

For this example, let us pick $\dot{x} = 0$. Then Eqs. 6.4 reduce to a statement that the difference flux values corresponding to left and right states is zero. Consider the flux values given in Table 6.1 below ($\gamma = 1.4$).

Quantity	Value
Energy Flux $\frac{\gamma pu}{\gamma - 1} + \rho u^3/2$	14.9085
Mass Flux ρu	2.36643
Momentum Flux $p + \rho u^2$	6.6

Table 6.1 Flux values for 1-d steady shock

Two sets of solutions are possible that satisfy the relations corresponding to values of the fluxes. These are shown as "Solution 1" and "Solution 2" in Table 6.2 below.

Quantity	Solution 1	Solution 2
pressure p	1.0	4.5
density ρ	1.0	2.66667
velocity u	2.36643	0.88741
Mach number M	2.0	0.57735
Entropy p/ρ^γ	1.0	1.13987
eigenvalue $(u - c)$	1.18322	−0.64963

Table 6.2 Possible left and right states

Solution 1 can either correspond to q_l or q_r and so can Solution 2. When Solution 1 or Solution 2 is selected to be both q_l and q_r, we obtain the trivial case with no discontinuity. Solution 1 corresponds to supersonic flow along the positive x direction

with $M_1 = 2.0$. Solution 2 corresponds to subsonic flow along the positive x direction with $M_2 = 0.57735$. When Solution 1 is taken to correspond to q_l and Solution 2 to correspond to q_r we have a shock-wave discontinuity. In this case, we note that there is an increase of entropy across the shock (when crossing over from supersonic to subsonic side) and we also note for later use that the eigenvalue $u - c$ (c is the local speed of sound) transitions from a larger value to a lesser value (in this case also from positive to negative value). If we specify that Solution 2 corresponds to q_l and Solution 1 to q_r, we have a so-called "expansion shock". The flow suddenly expands from subsonic to supersonic flow, entropy decreases, and the eigenvalue $u - c$ increases (from negative value to positive value in this case). The expansion shock can be ruled out as unphysical (but mathematically permissible) because of entropy decrease across the shock wave. This leaves us with the compressive shock wave as the one desirable solution. A geometric "entropy condition" will be discussed later as another means to avoid choosing expansion shocks as a possible solution component for the Riemann problem.

It is left as an exercise to the reader to show that "contact discontinuities" are also solutions to Eqs. 6.4.

$$\begin{aligned} \dot{x} &= u_l = u_r \\ p_l &= p_r \\ \rho_l &\neq \rho_r \end{aligned} \tag{6.5}$$

Substituting these values into Eqs. 6.4 will also show the appropriateness of the term "linearly degenerate" that is often used to describe the contact discontinuity. Shock waves are correspondingly referred to as "genuinely nonlinear".

Linear Constant-Coefficient Equations

Let us now apply the jump relations to a system of linear equations with $f(q) = Aq$, where A is a constant matrix. The jump relations become

$$-\dot{x}\,[q] + A\,[q] = 0 \tag{6.6}$$

The similarity between the above and the equation for the eigenvalue and eigenvector of A is obvious. Therefore $\dot{x}$ is an eigenvalue of A and $[q]$ is proportional to the corresponding right eigenvector.

Since Eq. 6.1 is assumed to be hyperbolic, $\partial f/\partial q = A$ has m real eigenvalues ($\lambda_i, i = 1,\ldots,m$) and a set of m linearly independent right eigenvectors ($r_i, i = 1,\ldots,m$). Each eigenvalue corresponds to one choice of discontinuity-propagation speed $\dot{x}$. The jump in dependent variables across that discontinuity is proportional to r_i (see Fig. 6.1). Let

$$[q]_i = \alpha_i r_i \tag{6.7}$$

where α_i is a coefficient of proportionality.

The transition from q_l to q_r takes place across the m discontinuities (λ_i need not all be distinct).

$$\sum_m \alpha_i r_i = q_r - q_l = \Delta q \tag{6.8a}$$

$$\text{or} \quad R\alpha = \Delta q \tag{6.8b}$$

where α is the m-vector of coefficients and R is the matrix whose columns are right eigenvectors. We can easily solve for α

$$\alpha = R^{-1}\Delta q \tag{6.9}$$

Hyperbolicity guarantees the existence of R^{-1}. Equation 6.9 can also be written as

$$\alpha = L\Delta q \tag{6.10}$$

where L is the matrix whose rows are left eigenvectors (taken in the same order as the right vectors of R) normalized such that

$$LR = RL = I \tag{6.11}$$

where I is the identity matrix.

Once again, just as in the nonlinear case, we see that each transition is a one-parameter family of solutions. The most convenient parameter choice is α_i. Of course, an appropriate element of q_{r_i} (using the notation that there is a q_{r_i} and q_{l_i} for each transition i) can be used. The corresponding element of r_i must not be zero for solvability.

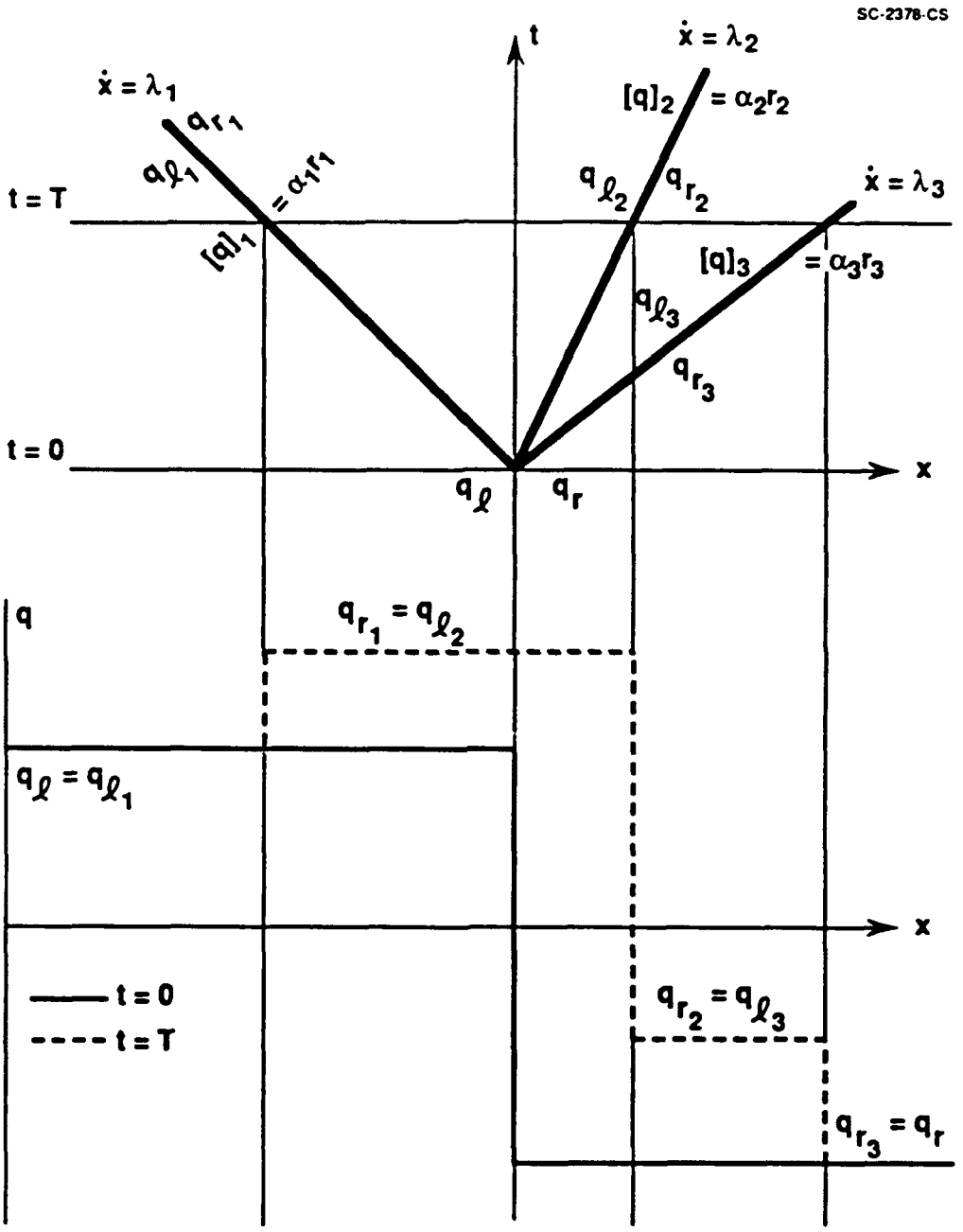


Figure 6.1 Riemann solver for linear system of equations

We clearly see in the case of linear systems of equations that the transition from q_l to q_r is made up of a sequence of constant states separated by transitions. The m unknown free parameters are chosen in such a fashion that the sum of the m transitions matches the total change from q_l to q_r.

Across Discontinuities and Along Characteristics

A few words comparing the jump relations *across* discontinuities and invariance of certain variables *along* characteristics are in order. First, we note that for linear equations, the characteristic speeds and discontinuity-propagation speeds are one and the same. Premultiplying Eq. 6.1 by l_i, we see that

$$l_i\left(\frac{\partial q}{\partial t} + A\frac{\partial q}{\partial x}\right) = 0 \tag{6.12a}$$

$$l_i\frac{\partial q}{\partial t} + \lambda_i l_i\frac{\partial q}{\partial x} = 0 \tag{6.12b}$$

$$\frac{\partial \alpha_i}{\partial t} + \lambda_i\frac{\partial \alpha_i}{\partial x} = 0 \tag{6.12c}$$

where $\alpha_i = l_i q$. Along lines with $\dot{x} = \lambda_i$,

$$\begin{aligned}\frac{d\alpha_i}{dt} &= \frac{\partial \alpha_i}{\partial t} + \frac{dx}{dt}\frac{\partial \alpha_i}{\partial x} \\ &= \frac{\partial \alpha_i}{\partial t} + \lambda_i\frac{\partial \alpha_i}{\partial x} = 0\end{aligned} \tag{6.13}$$

This shows that the variables α_i are constant along the characteristics whose slope is given by $\dot{x} = \lambda_i$ (same as the shock speed).

Riemann Invariants Along Characteristics

The α_i are called Riemann invariants. In the linear case we can clearly see that since α_i are constant along the characteristics directions (also shocks), if they were discontinuous across these lines to begin with, the left value will be maintained constant on the left side of the characteristic (shock) and the right value will be maintained constant on the right side of the characteristic (shock). Therefore the

initially present discontinuities across the characteristic (shock) will be preserved along these lines. For the Riemann problem associated with linear equations, no continuous transition from q_l to q_r is possible.

Continuous Transitions

We have now discussed the case of nonlinear jumps (Eq. 6.3) and linear discontinuities (Eq. 6.6). In order to complete the picture, we must consider the case of continuous transition for the nonlinear problem. One must consider this because when $f = f(q)$ and $\partial f/\partial q = A(q)$, the eigenvalues are functions of q also. Let us imagine that a change between state q_{l_i} to q_{r_i} (i.e. the i-th transition) is made up of a sequence of incremental changes, with each increment described by a linear equation with a local value of constant coefficient matrix A. This situation is depicted in Fig. 6.2a in two ways. First, the local jump (for the i-th transition) is shown in the $q - x$ plane. The sequence of incremental changes are shown by the indices $0 - 9$. These values are next shown in a state space diagram ($q - \alpha$ plane).

When the actual equations being considered are linear, the local characteristic and discontinuity speeds for each incremental state is identical. For nonlinear equations, the local incremental characteristic (and discontinuity) directions are dependent on the local q. One can have either convergent or divergent characteristic directions (Fig. 6.2b) as we sweep through the incremental states.

The case of divergent characteristics is shown on the top figure. The characteristics are marked by the notation λ_{ij} where i denotes the i-th transition and j denotes the incremental transition between incremental states $j - 1$ and j. In the case of divergent characteristics, λ_{i1} is to the left of λ_{i9} in the $x - t$ plane. The local incremental left and right states corresponding to these incremental characteristics are also marked for each characteristic. This picture makes complete physical sense and results in the i-th transition being comprised of a series of incremental and small jumps which are single-valued in the $x - t$ plane.

When we evaluate $\lambda_i(q_{l_i})$ and $\lambda_i(q_{r_i})$, we may find that they are convergent. If we assume that the i-th transition is continuous in this case, we obtain the situation depicted in the lower figure in Fig. 6.2b. The transition between incremental states

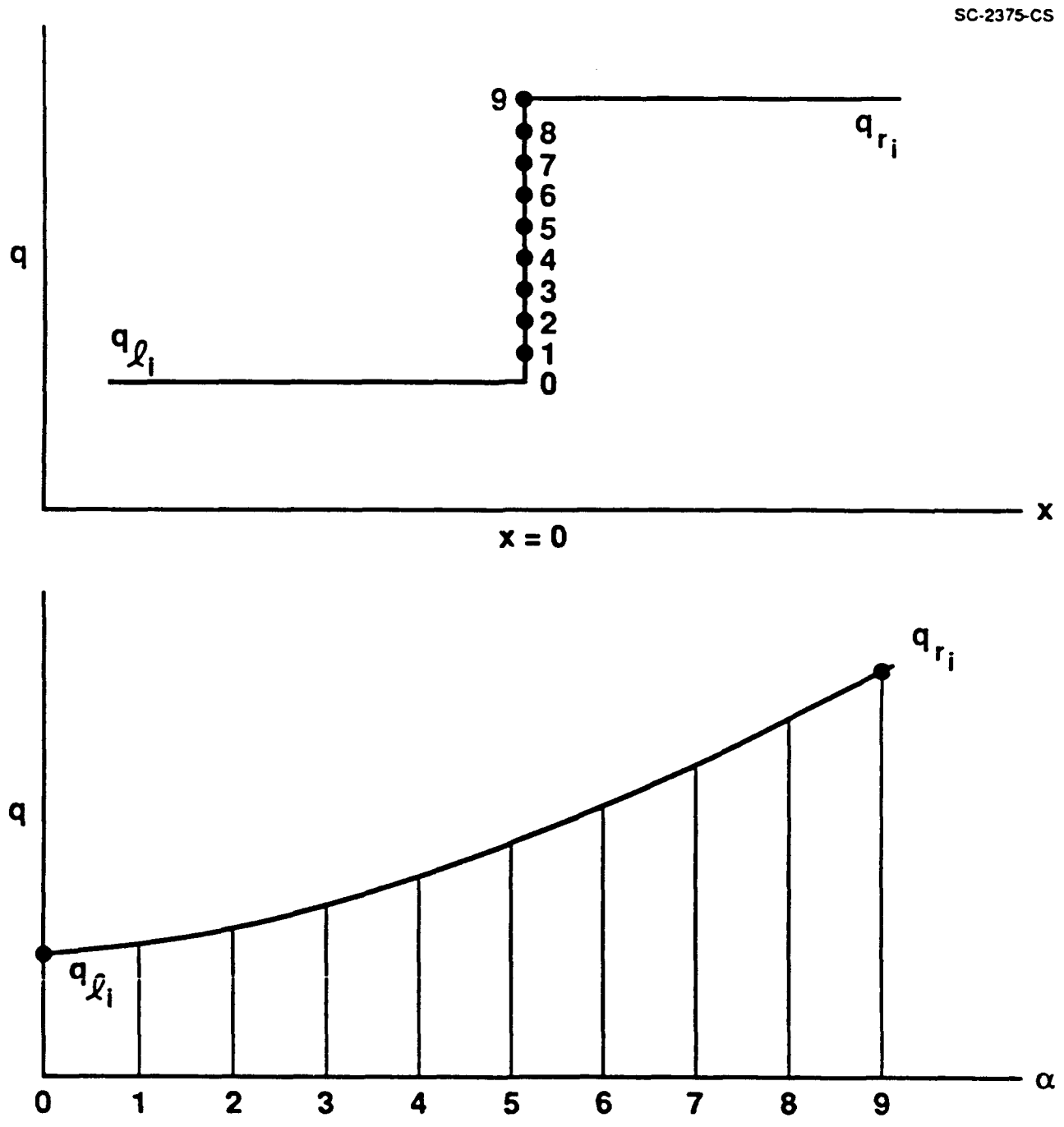


Figure 6.2a State space setup for continuous transition

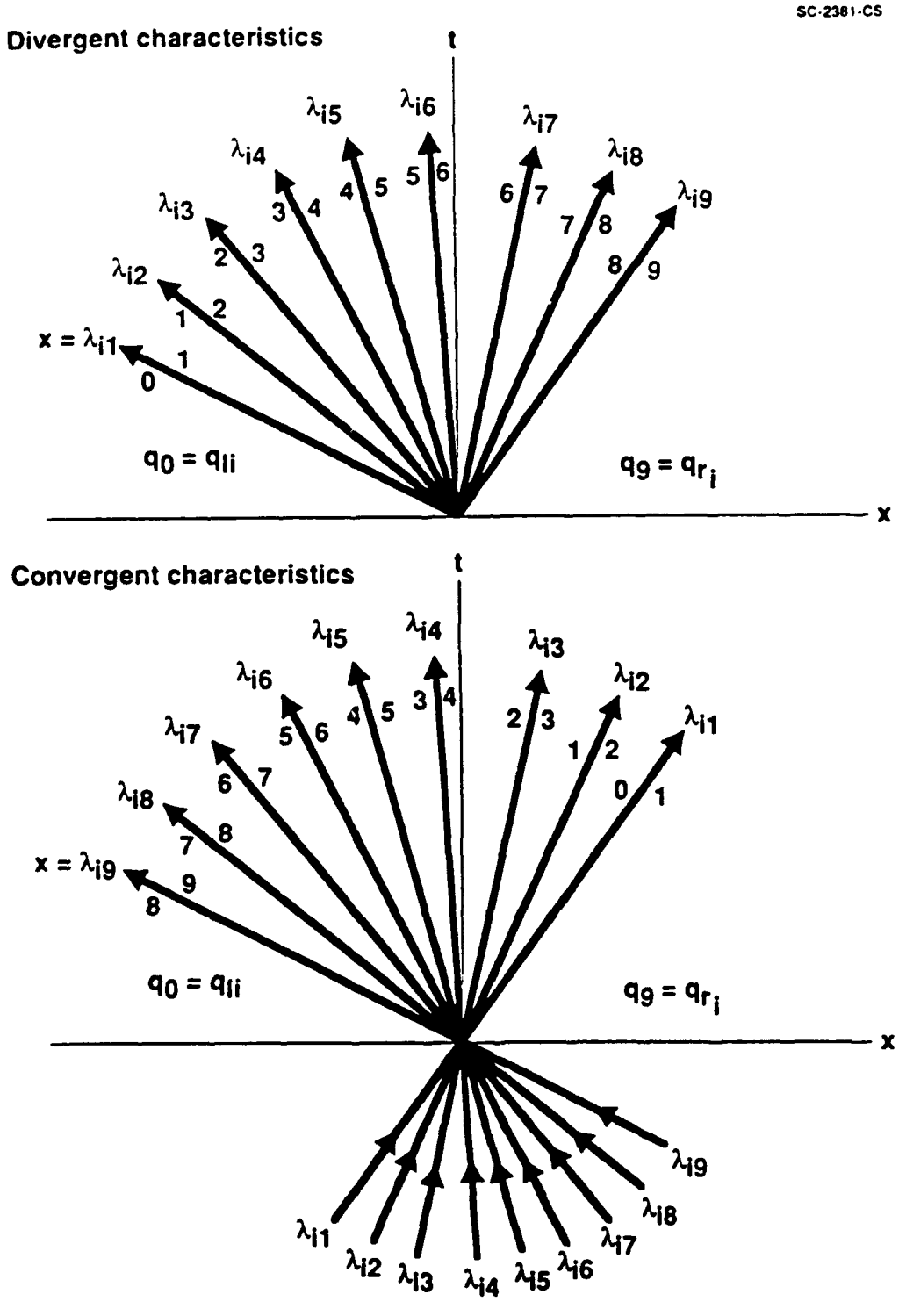


Figure 6.2b Physical and nonphysical continuous transitions

$0-1$ is by definition to take place across the characteristic identified by λ_{i1}. The transition between incremental states $1-2$ is to take place across λ_{i2}, etc. Looking at the situation depicted in the figure, the physical absurdity of the assumption of continuous incremental transitions is obvious. Each of the incremental states is to be found at two values of x/t in the diagram. Therefore, in order to avoid physically absurd transitions, the case of convergent characteristics must not be associated with continuous transitions but must be coupled to the discontinuous transition case that has already been discussed earlier. However, a continuous transition for the case of convergent characteristics can exist along the negative-t direction (the lower half of the $x-t$ plane). This indicates that convergent characteristics result in shock waves.

One may also consider a combination of convergent and divergent characteristic directions in the same i-th transition. In the part with the convergent characteristics, the solution will be multivalued and therefore we must rule out this case. Later, we will discuss the concept of a geometric entropy condition related to this. Diverging characteristics lead to single valued transitions between incremental states and are allowed to occur. Thus we see that for genuinely nonlinear problems, continuous "rarefactions" (with divergent characteristics) may arise.

Having said all of the above regarding the physical absurdity of the combination of convergent characteristics and smooth transitions, we will see later (during the discussion of the Osher scheme) that we will indeed choose to use such a mathematically possible approach to define an approximate Riemann solver. We will leave the details until then.

Let us now consider the mathematics of continuous transitions. By analogy with the linear case where Eq. 6.7 was valid, we can think of α_i as a running parameter over the incremental changes in the sequence of locally linear incremental continuous transitions (in state space) associated with the i-th characteristic field. Thus

$$\left(\frac{dq}{d\alpha}\right)_i = r_i(q) \tag{6.14}$$

Across such a continuous transition,

$$\begin{aligned} q_{r_i} - q_{l_i} &= \int_0^{\alpha_i} \frac{dq}{d\alpha} d\alpha \\ &= \int_0^{\alpha_i} r_i(q(\alpha)) d\alpha \end{aligned} \tag{6.15}$$

Once again we see the role of α_i as the free parameter, in this case the running parameter too.

Riemann Invariants Across Continuous Transitions

We saw that in the linear case, α_i can be shown to be the Riemann invariants along characteristics. In the nonlinear case, Eqs. 6.12a and 6.12b remain valid (even though $A = A(q)$ and not a constant matrix). However, the transition from Eq. 6.12b to Eq. 6.12c is only possible for linear equations and for nonlinear equations, in general, when there are no more than two dependent variables[36]. Therefore the existence of Riemann invariants *along* characteristics is limited to linear equations and, in general, to a set of two nonlinear equations.

In contrast to the above, Riemann invariants across continuous transitions exist for the case of any number of nonlinear systems of conservation laws. However, deriving them may be very difficult, depending on the complexity of the equations. We already have seen in Eq. 6.14 that the change in dependent variables across the i-th (continuous nonlinear) transition (rarefactions) is in the direction of the local right eigenvector corresponding to the i-th eigenvalue of the Jacobian matrix A. Since the set of right eigenvectors and left eigenvectors is orthogonal (Eq. 6.11), we see that

$$l_j \left(\frac{dq}{d\alpha}\right)_i = 0 \quad \text{for} \quad i \neq j \tag{6.16}$$

Therefore there are $m - 1$ directions that are orthogonal to the direction in state space given by r_i. This is always true but here we relate the orthogonal directions to the left eigenvectors. Now let us consider variables ψ which is constructed to obey

$$\nabla_q \psi^i \cdot r_i(q) = 0 \tag{6.17}$$

Such variables can exist because by inspecting Eq. 6.16, we see that the gradient of ψ can be combinations of the left eigenvectors that are orthogonal to r_i. Such variables are invariant *across* the transition in state space (and across the divergent characteristics representing the smooth transition in physical space) because

$$\begin{aligned} \frac{d\psi^i}{d\alpha_i} &= \nabla_q \psi^i \cdot \frac{dq}{d\alpha_i} \\ &= \nabla_q \psi^i \cdot r_i = 0 \end{aligned} \tag{6.18}$$

From Eqs. 6.17 and 6.18 we see that for each transition, there can be only $m-1$ Riemann invariants. These $m-1$ invariance relations provide $m-1$ equations for solving for m unknowns (say q_r) and *once* again one suitable parameter must be chosen to be the free parameter for the transition. For more details regarding Riemann invariants across transitions the reader is referred to Refs. 15 and 16.

Complete Solution of Riemann Problem

Now we are ready to put together the complete picture of the exact solution to the one-dimensional Riemann problem. The solution is made up of piecewise-constant states separated by m transitions associated with the m eigenvalues of the Jacobian matrix $\partial f/\partial q$. The transitions may be continuous (rarefactions) or discontinuous. Across each discontinuous transition, the jump relations (known as the Rankine-Hugoniot relations for the Euler equations) hold. Each transition, irrespective of type, is a one-parameter transition. There are therefore m unknown parameters. These m values must be chosen so that the sum of the corresponding transitions adds up to the total change $q_r - q_l$.

When all the transitions are known, the various discontinuity speeds and the smallest and largest characteristic speeds associated with each continuous transition are known. Knowing these one can, in particular, determine what the dependent variables are along the line $\dot{x} = 0$. We denote this solution as q^R

Expansion Shocks and Geometric Entropy Condition

We saw in the earlier example with the one-dimensional Euler equations that among the multiple solutions that satisfy the Rankine-Hugoniot relations, one corresponds to an expansion shock. Across such a discontinuity,

$$\lambda_i(q_{l_i}) < \lambda_i(q_{r_i}) \quad . \tag{6.19}$$

In the case shown in Tables 6.1 and 6.2, the eigenvalue considered is $(u-c)$. Across such an expansion shock, the entropy decreases. For the physically correct shock

wave, the entropy should increase and for the example being considered we also see that

$$\lambda_i(q_{l_i}) > \lambda_i(q_{r_i}) \quad . \tag{6.20}$$

We also saw, in our discussion of continuous transitions, that the assumption of continuous transition with single-valued solutions over the transition is inconsistent with $\lambda_l > \lambda_r$ (overturned characteristics arise). While the "entropy"-based method of ruling out expansion shocks is possible for the Euler equations where a physical entropy variable is available, the geometric condition for selection of discontinuous or continuous transitions is useful for all hyperbolic systems of conservation laws. This geometric "entropy condition" may be stated as follows. For the i-th transition, select discontinuity relations if $\lambda_i(q_{l_i}) > \lambda_i(q_{r_i})$ and select the continuous transition otherwise.

Integral Form for 1-D Conservation Laws

We now specialize the general multidimensional formulation of Section 4 to the case of one-dimensional conservation laws in order to help the reader see many algorithmic aspects come together without the added complexity of multidimensional geometry formulation.

Beginning with Eq. 6.1 and integrating with respect to x and t, we obtain

$$\int_t \int_x (q_t + f(q)_x)\, dx\, dt = 0 \tag{6.21}$$

$$\int_x \left(\int_t q_t\, dt \right) dx + \int_t \left(\int_x f_x\, dx \right) dt = 0 \tag{6.22}$$

$$(\bar{q}_j^{n+1} - \bar{q}_j^n)\Delta x + (\bar{f}_{j+1/2}^n - \bar{f}_{j-1/2}^n)\Delta t = 0 \tag{6.23}$$

where t^n, t^{n+1} and $x_{j+1/2}$, $x_{j-1/2}$ define the limits of integration,

$$\bar{q}_j = \frac{1}{\Delta x} \int_{x_{j-1/2}}^{x_{j+1/2}} q\, dx \tag{6.24a}$$

is the cell average of the dependent variables and

$$\bar{f}_{j\pm 1/2} = \frac{1}{\Delta t} \int_{t^n}^{t^{n+1}} f\, dt \tag{6.24b}$$

is the average flux along cell boundaries over an interval of time.

Equation 6.23 is the fully discrete integral form of the 1-d system of conservation laws presented as Eq. 6.1 and can also be written as

$$\frac{\bar{q}_j^{n+1} - \bar{q}_j^n}{\Delta t} + \frac{\bar{f}_{j+1/2}^n - \bar{f}_{j-1/2}^n}{\Delta x} = 0. \tag{6.25}$$

Even though Eq. 6.25 resembles a finite-difference formula, it must be noted that it is an exact relation that must be satisfied by any exact solution of the differential equations. The integral form of the equations does not demand the existence of derivatives but only weaker conditions of integrability and solutions of Eq. 6.25 can also therefore include "weak solutions."

The semi-discrete version of Eq. 6.25 can be written as

$$\frac{\partial \bar{q}}{\partial t} + \frac{\hat{f}_{j+1/2} - \hat{f}_{j-1/2}}{\Delta x} = 0 \tag{6.26}$$

and can either be obtained by integrating Eq. 6.1 only with respect to x or by taking the limit $\Delta t \to 0$ in Eq. 6.25. In Eq. 6.26, $\hat{f}_{j\pm1/2}$ is the flux at cell boundary $x_{j\pm1/2}$.

One-Dimensional Numerical Methods

We observed in the last subsection that Eq. 6.23 resembled a finite-difference formula that may be used to advance the cell averages $\bar{q}$ from one time level to the next if the cell boundary (face) values of the fluxes can be defined. The key is being able to obtain the cell face values of the flux from known values of $\bar{q}$. This may be accomplished using piecewise polynomial interpolation (known as reconstruction) as explained in Section 4.

Consider the initial value problem for Eq. 6.23 defined by adding to that equation the initial conditions $\bar{q}_j^0, j = 1, \cdots, J$. Then, a numerical algorithm to solve Eq. 6.23 may be defined as follows:

a) Interpolate $\bar{q}_j$ to obtain piecewise polynomial pointwise behavior of q within each cell.

b) Each polynomial (within each cell) may be evaluated at $x_{j\pm1/2}$ for that cell. Collecting all such values, we find that we have, at each cell face, left and right values $(q_L, q_R)_{j+1/2}$.

c) Resolve the discontinuity at each cell face using solutions to the Riemann problem. (The solution procedure is usually referred to as the Riemann Solver.) This will result in a knowledge of $\hat{f}_{j+1/2}$ which we shall henceforth call the numerical flux.

d) Substitute $\hat{f}_{j\pm1/2}$ into Eq. 6.23 to advance the solution to the next time level. Proceed to step (a) and repeat.

Notes:

1) In step (a), we must construct piecewise polynomials that match the given cell averages. This is different from the usual interpolation of discrete pointwise values. There are at least three different ways of performing such interpolation in order to reconstruct the pointwise behavior of the original dependent variables in each cell — i) reconstruction by deconvolution (RD), ii) reconstruction using the primitive function formulation (RP), and iii) reconstruction by matching cell averages directly (RM). The reconstruction procedure outlined in Section 4 of this report is based on (RM). A discussion of (RD) and (RP) can be found in Reference 3 and papers cited therein.

2) One may wonder why piecewise polynomials should be used, especially when one sees in step (b) that this will lead to discontinous behavior at cell interfaces. In fact, the choice of piecewise polynomials is particularly apt for just that reason. After all, we must allow our interpolation model to permit discontinuities since our goal is to be able to compute "weak" solutions. It is true, however, that with piecewise polynomials of the type described in this report, the approximation always pushes any discontinuity to be at the cell interfaces. Further refinements have already been devised to enable discontinuities to be located even within the cell ("subcell resolution" — Ref. 37) but it is beyond the scope of this report to delve into such advances.

3) When dealing with systems of equations, questions arise regarding the choice of variables to interpolate: should the reconstruction techniques be based on match-

ing the basic conservation variables' averages, "primitive" variables', "characteristic" variables', etc.? These issues are not considered in this report but are covered in Refs. 3 and 4.

4) In the case of interpolation with piecewise-constant polynomials, if we consider two neighboring cells, we have two sets of constant values, one to the left of and one to the right of each cell interface. This resembles the classical Riemann problem. When higher degree polynomials are chosen, the left and right states are not constant but a Riemann Solver may still be used to construct the solution at the instant of initial contact between the discontinuities. More sophisticated Riemann Solvers may also be sought — those that resolve piecewise linear left and right state variations, etc. In this report, the semi-discrete formulation is utilized to construct higher-order time-accurate schemes. It may be observed by looking at Eq. 6.26 that if we use a method-of-lines approach and embed the semi-discrete form in a Runge-Kutta time-integration scheme, for example, then only the pointwise values of the cell interface fluxes are required. These can be obtained using a Riemann Solver based on local values of left and right states.

5) Going back to Note 2, we can also add that for smooth data, the magnitude of the difference between q_l and q_r behaves with $O(\Delta x^{r+1})$ where r is the degree of the interpolating polynomial. Thus, the piecewise polynomial approach is appropriate for obtaining both smooth solutions and solutions with discontinuities. For smooth solutions, the need for using "good" Riemann Solvers becomes decreasingly important with increasing degree of polynomial approximation.

6) The procedure for multidimensional flows is similar to that for one-dimensional problems. Interpolation in step (a) must be carried out in a suitable multidimensional way. We saw in Section 4 that the boundary integration of Eq. 4.22 can be replaced by a suitable quadrature. At each quadrature point there are two sets of values of the dependent variables, one set from the left cell and the other set from the right cell (or the inside and outside cells as the perspective may be). If we have used piecewise-constant interpolation, we have a local Riemann problem (piecewise-constant states extending to a finite distance away from the discontinuity placed at the cell face) in the direction of the local normal to the common cell face. With higher-degree interpolation, the remarks of Note 4 can

be brought to bear: for an instant of time just after the left and right states are allowed to interact, the solution to the one-dimensional Riemann problem (along the direction that is normal to the cell face) is applicable. Therefore the need for a multidimensional Riemann Solver and a Riemann Solver that can deal with non-piecewise-constant left and right states can be obviated by exploiting a suitable combination of several pointwise (in time and space) Riemann problems.

We now use a set of figures to help visualize the above concepts. Figure 6.3 simply outlines the integration limits for the 1-d integral form. Figure 6.4 shows how an initial value problem (IVP) for Eq. 6.1 can be replaced by the corresponding one for cell averages given by Eq. 6.25. Assuming that piecewise constant reconstruction was used, Figure 6.5 zooms in on one local Riemann problem (IVP with piecewise constant states) and Figure 6.6 helps visualize how the individual Riemann problems, taken together, provide the means to update the cell averages to the next time level. Figure 6.7 shows the use of piecewise-constant and piecewise-linear interpolation in one spatial dimension.

Using the Riemann Solver

We have already seen that the exact solution to the Riemann problem is made up of piecewise constant states separated by transitions. Each transition is associated with an eigenvalue of the Jacobian matrix. For the 1-d Euler equations, there are three eigenvalues $u - c$, u and $u + c$, where c is the speed of sound ($c = \sqrt{\gamma p/\rho}$). The transitions associated with $u \pm c$ can either be a shock wave or a rarefaction and that associated with u is called a contact discontinuity. The following section on the Godunov scheme provides formulae for the construction of the exact solution. In particular, this provides the extents of the piecewise constant states and the magnitudes of the transitions. From this information, the value of q along the ray $\theta = 0$ may be determined. We denote this by q^R and the corresponding flux as $f^R = f(q^R)$.

Consider now Eq. 6.26 and the steps (a)-(d) of the solution procedure given in the previous subsection. We assume piecewise constant behavior of dependent variables

$$q(x) = \tilde{q}_j, \quad x_{j-1/2} < x < x_{j+1/2} \tag{6.27}$$

$$\left(\bar{q}_j^{\,n+1} - \bar{q}_j^{\,n}\right)\Delta x + \left(\hat{f}_{j+1/2} - \hat{f}_{j-1/2}\right)\Delta t = 0$$

$$\frac{\bar{q}_j^{\,n+1} - \bar{q}_j^{\,n}}{\Delta t} + \frac{\left(\hat{f}_{j+1/2} - \hat{f}_{j-1/2}\right)}{\Delta x} = 0$$

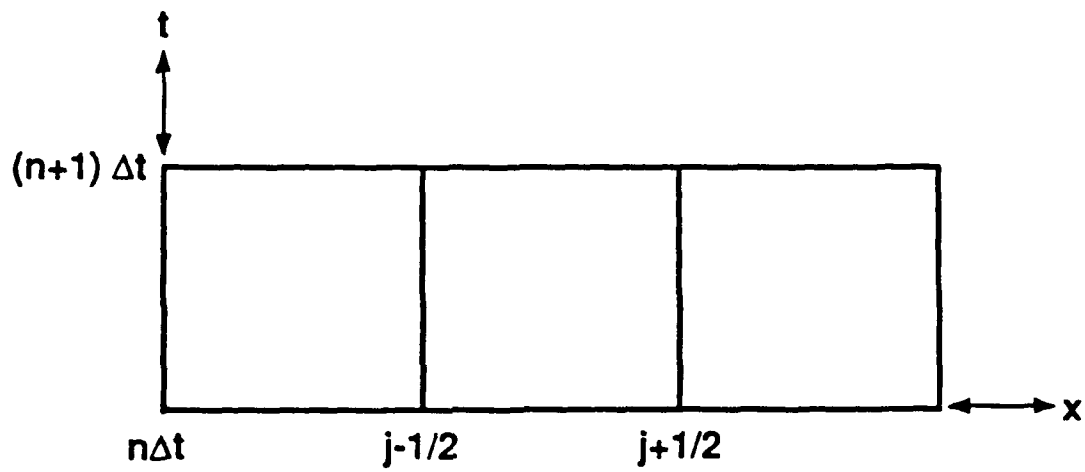


Figure 6.3 1-d integration cell limits

INITIAL VALUE PROBLEM

$$q_t + f_x = 0$$

$$q(0,x) = q_0(x) \quad \text{INITIAL VALUE}$$

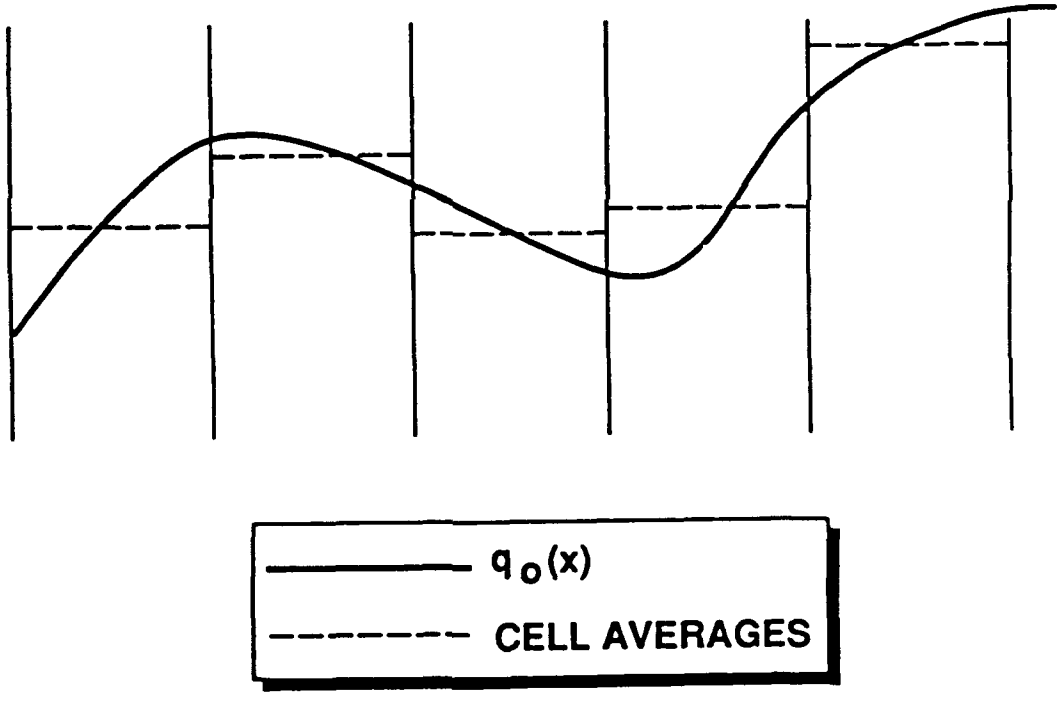
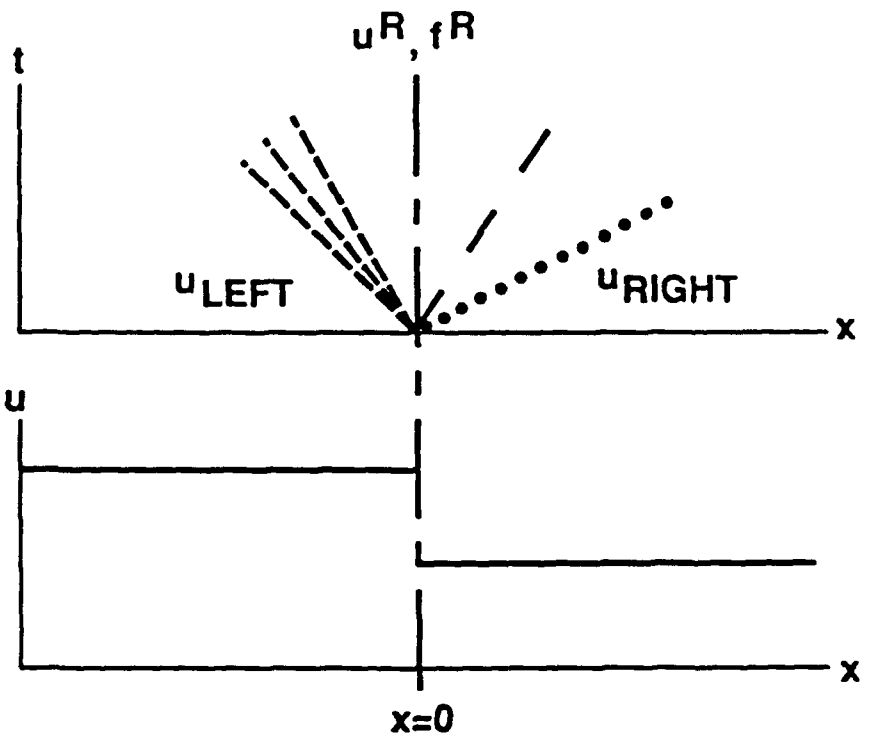


Figure 6.4 IVP for cell averages

RIEMANN SOLVER



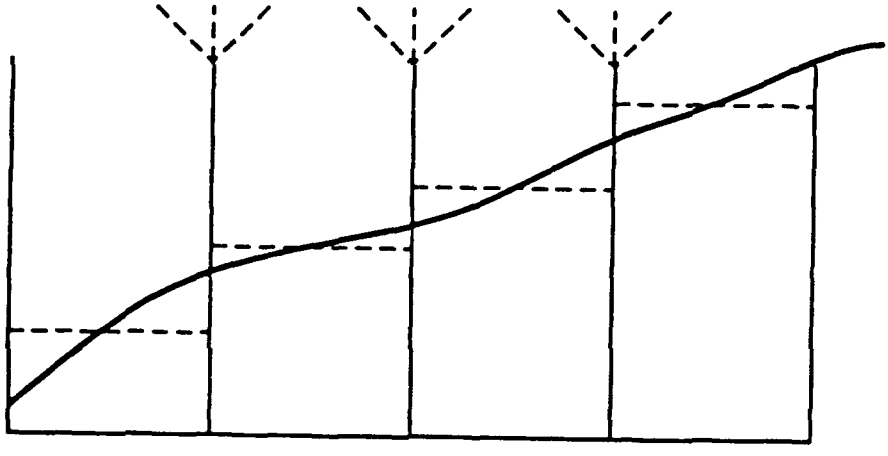
f^R = FLUX AT- $x_{j+1/2, t=0+\varepsilon}$

f IS OBTAINED BY USING
A KNOWLEDGE OF DECOMPOSITION
OF INITIAL DISCONTINUITIES

Figure 6.5 Riemann Solver

USING RIEMANN SOLVER

$$\frac{\bar{q}^{n+1} - \bar{q}^{n}}{\Delta t} + \frac{\hat{f}_{j+1/2} - \hat{f}_{j-1/2}}{\Delta x} = 0$$



WE DEFINE $\hat{f} = f^{RIEMANN}$

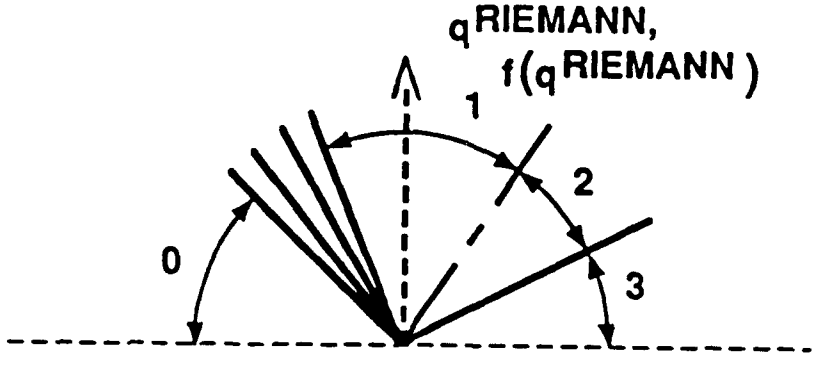


Figure 6.6 Using Riemann problem solutions

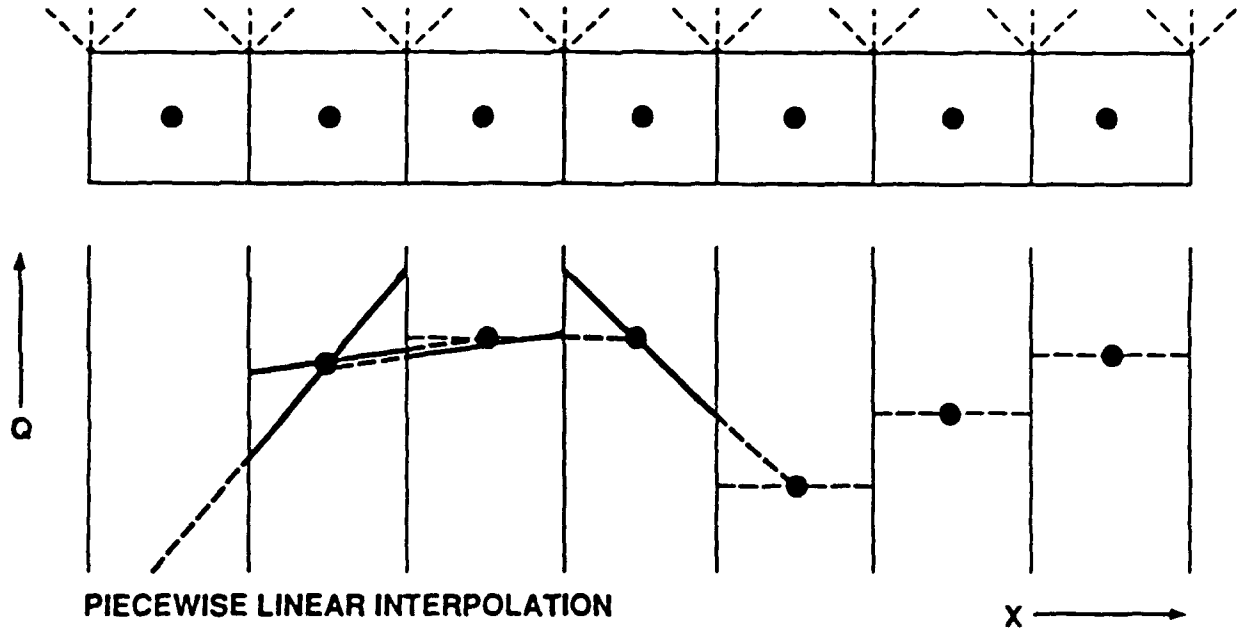


Figure 6.7 Piecewise constant and linear interpolations

This results in local Riemann problems which can be solved to construct

$$\hat{f}_{j+1/2} = f^R_{j+1/2} \tag{6.28}$$

These numerical fluxes can be substituted into Eq. 6.26 along with a suitable time-stepping procedure to advance $\bar{q}_j$.

Since for hyperbolic systems it takes finite time for spatially separated locations to influence each other, for a sufficiently small time step, the Riemann problem from $x_{j-1/2}$ will not affect the solution to the Riemann problem at $x_{j+1/2}$ and vice-versa. We call such a time step as Δt_{CFL} with the subscript referring to the Courant-Friedrichs-Lewy stability limit for linear equations.

$$\text{For} \quad \Delta t \leq \Delta t_{CFL}, \quad \bar{f}^n_{j+1/2} = \hat{f}_{j+1/2} \tag{6.29}$$

and therefore the fully discrete form, Eq. 6.25 may also be used to advance the solution $\bar{q}_j$.

We must note that, when there are source terms in 1-d, for higher-order polynomial interpolation, and for multidimensional problems, it is still true that two spatially separated Riemann problems will not influence each other for a sufficiently small interval of time. However, each Riemann problem solution is no longer self similar under these circumstances and therefore Eq. 6.29 is not true. In such cases, it is convenient to resort to the method of lines (semi-discrete) approach.

We now illustrate three properties:

1) Discretization methods such as those described in this section (which use a Riemann Solver) are "upwind" schemes.

2) Methods based on piecewise constant interpolation are only first-order accurate.

3) First-order accurate upwind schemes are monotonocity preserving.

When a Riemann flux is used as the cell interface flux, the fully discrete integral form defined in Eq. 6.25 becomes

$$\bar{q}^{n+1}_j = \bar{q}^n_j - \frac{\Delta t}{\Delta x}(\hat{f}^n_{j+1/2} - \hat{f}^n_{j-1/2}) \tag{6.30}$$

where $\hat{f}_{j+1/2} = f^R_{j+1/2}$. Adding and subtracting f_j from the second term on the right hand side (RHS), we can rewrite that term to be

$$\hat{f}_{j+1/2} - \hat{f}_{j-1/2} = (\hat{f}_{j+1/2} - f_j) + (f_j - \hat{f}_{j-1/2}) \tag{6.31}$$

The term $\hat{f}_{j+1/2} - f_j$ includes the effect of all left-moving waves from the right. The term $f_j - \hat{f}_{j-1/2}$ includes the effect of all right-moving waves from the left. Therefore, Eq. 6.30 describes a method of updating $\bar{q}_j$ that accounts for the appropriate signal propagation effects, and hence describes an "upwind" scheme.

We leave it to the reader to show that when $a > 0$ for the one-dimensional version of the linear wave equation corresponding to Eq. 5.3,

$$\hat{f}_{j+1/2} = a\bar{u}_j, \quad \hat{f}_{j-1/2} = a\bar{u}_{j-1} \tag{6.32}$$

and therefore, we obtain

$$\bar{u}^{n+1}_j = \bar{u}^n_j - a\frac{\Delta t}{\Delta x}(\bar{u}^n_j - \bar{u}^n_{j-1}) \tag{6.33}$$

Once again, we see that the numerical algorithm defined using piecewise constant polynomials and a Riemann Solver results in an "upwind" scheme.

For piecewise constant and piecewise linear polynomial approximations, the value of the cell average is also the pointwise value at the midpoint of each cell. Thus, rewriting Eq. 6.33 as

$$u^{n+1}_j = u^n_j - \nu(u^n_j - u^n_{j-1}) \tag{6.34}$$

where $\nu = a\,\Delta t/\,\Delta x$, we see that

$$u^{n+1}_j = (1-\nu)u^n_j + \nu u^n_{j-1} \tag{6.35}$$

and therefore the method is monotonicity preserving for the linear wave equation as long as $\nu \leq 1$. It is clear that the values u^{n+1}_j will be bounded by the maxima and minima of u^n_j, and if the u^n_j described a monotone profile, then u^{n+1}_j will preserve such monotonicity. A Taylor-series analysis of Eq. 6.34 will also show that the finite-difference scheme is first-order accurate.

As motivation for the following sections, we look at Eq. 6.35 from the following perspective. For the linear wave equation, the solution u_j^{n+1} should be equal to the solution at $t = t^n$ at the foot of the characteristic drawn backwards from x_j, t^{n+1}. The RHS of Eq. 6.35 is equal to that value computed using linear interpolation of the discrete values u_j^n and u_{j-1}^n.

We now describe some generalizations that we can use as framework for describing many schemes including those that are based on "approximate" Riemann Solvers or even those that are not based on Riemann Solvers at all. Extending the second term on the RHS of Eq. 6.30 even further along the lines shown in Eq. 6.31,

$$\hat{f}_{j+1/2} = \frac{f(q_r)_{j+1/2} + f(q_l)_{j+1/2}}{2} - \frac{(f(q_r)_{j+1/2} - f^R_{j+1/2}) - (f^R_{j+1/2} - f(q_l)_{j+1/2})}{2} \tag{6.36}$$

Note the use of superscript R to denote the Riemann problem solution and the subscripts r and l to denote right and left states. This can be rewritten, after dropping the subscript $j + 1/2$, as

$$\hat{f} = \frac{f(q_r) + f(q_l)}{2} - \frac{(\Delta f)^+ - (\Delta f)^-}{2} \tag{6.37}$$

where the terms with the + and − superscripts represent the net flux difference across various discontinuities and continuous transitions grouped by "positive" and "negative" directions, respectively. It is shown in Ref. 13 how this form can be used to represent methods using Osher's or Roe's approximate Riemann Solvers in addition to that using the exact Riemann Solver described earlier in this section (also known as the Godunov scheme). In fact, Eq. 6.37 can be used to represent even Split-Flux schemes as well as schemes that do not use Riemann Solvers at all. For example,

$$\hat{f} = \frac{f(q_r) + f(q_l)}{2} - \phi\frac{(q_r - q_l)}{2} \tag{6.38}$$

where ϕ can be a positive constant following the Lax-Friedrichs scheme or computed as the absolute value of the maximum local eigenvalue in the manner of the Rusanov scheme. We have already observed that such simpler approaches become quite useful with higher degree polynomial interpolation. These ideas will be revisited in somewhat more detail below.

Operational Unification

At the surface, the Riemann problem and its solution may seem to be limited in its applicability to one-dimensional problems with piecewise constant states. In fact, the one-dimensional Riemann Solver can be applied effectively to multidimensional problems as well as higher-order polynomial interpolation. In the earlier subsection on "One-Dimensional Numerical Methods," Note 4 dealt with the usefulness of the Riemann Solver when piecewise-linear and higher-degree interpolation (reconstruction) polynomials are used. Note 6 commented about multidimensional problems.

The previous subsection discussed how the one-dimensional Riemann solver is coupled to the one-dimensional integral law form of the equations to result in a numerical method. In this subsection, we first outline the framework of how one-dimensional Riemann Solvers may be applied in the direction normal to a multidimensional cell face. Next we develop an operational unification that will allow us to present the exact Riemann Solver along with approximate Riemann Solvers (Osher, Roe, Harten-Lax) and even the Rusanov and Lax-Friedrichs schemes (which have little to no significance as a Riemann Solver) within the same algebraic framework. This type of unification was first presented in Ref. 13. In the following subsections, we provide the appropriate algebraic details for all these schemes.

In previous subsections we used the subscripts l and r to define the left and right initial states of the Riemann problem and used l_i and r_i to define the local left and right states corresponding to the i-th transition. In this and subsequent subsections, it is more convenient to adopt a slightly different (but equally clear, we hope) nomenclature that will be explained along the way.

As before, let q^R be the exact solution to the local Riemann problem at a given cell-face quadrature point (these points were introduced in Section 4). We define the corrsponding flux (see Eq. 4.20) that includes the effect of the cell-face normal and surface area to be

$$F^R = \vec{F}(q^R) \cdot \hat{n}\, S \tag{6.39}$$

where $\hat{n}$ is the local unit outward-pointing normal, S includes that fraction of the cell-face area that can be attributed to the quadrature point based on its assigned

"weight", and $\vec{F}$ is the vector flux (Eq. 4.5). In a similar fashion, let us define

$$F = \vec{F}(q) \cdot \hat{n}\, S \tag{6.40a}$$

$$F_l = \vec{F}(q_l) \cdot \hat{n}\, S \tag{6.40b}$$

$$F_r = \vec{F}(q_r) \cdot \hat{n}\, S \tag{6.40c}$$

$$F_? = \vec{F}(q_?) \cdot \hat{n}\, S \tag{6.40d}$$

where "?" stands for any subscript.

Let us define dF_i^+ as the change in flux F across that part of the i-th transition that spans the positive part of the $\hat{n} - t$ plane (like the $x - t$ plane in one spatial dimension). Similarly, dF_i^- is the change in flux F across that part of the i-th transition that spans the negative part of the $\hat{n} - t$ plane. We note here that $dF_i^+ = 0$ over a transition for which all relevant wave speeds are negative, and vice-versa. For a transition having both positive and negative wave speeds, both dF_i^+ and dF_i^- may be nonzero.

Let us also define

$$\Delta q = q_r - q_l \tag{6.41a}$$

$$\Delta F^+ = \sum_{i=1}^{m} dF_i^+ \tag{6.41b}$$

$$\Delta F^- = \sum_{i=1}^{m} dF_i^- \tag{6.41c}$$

$$|\Delta F| = \Delta F^+ - \Delta F^- \tag{6.41d}$$

We can then define

$$F^R = F(q^R) \quad . \tag{6.42}$$

If we move to q^R from q_l, we can write

$$F^R = F(q_l) + \sum_{i=1}^{m} dF_i^- \quad . \tag{6.43}$$

If we approach q^R from q_r, we can write

$$F^R = F(q_r) - \sum_{i=1}^{m} dF_i^+ \quad . \tag{6.44}$$

Taking the average of Eq. 6.43 and Eq. 6.44, we get

$$F^R = \frac{(F_l + F_r)}{2} - \frac{1}{2}\left(\sum_{i=1}^{m} dF_i^+ - \sum_{i=1}^{m} dF_i^-\right) \quad . \tag{6.45}$$

This can be rewritten as

$$F^R = \frac{(F_l + F_r)}{2} - \frac{1}{2}(\Delta F^+ - \Delta F^-) \tag{6.46a}$$

or

$$F^R = \frac{(F_l + F_r)}{2} - \frac{1}{2}|\Delta F| \tag{6.46b}$$

The exact Riemann Solver can be expressed in all of the above forms. The approximate Riemann Solvers will only be expressible in the form of Eq. 6.43 to Eq. 6.46. The Rusanov and Lax-Friedrichs schemes will fit a modified form of Eq. 6.46 where the $|\Delta F|$ is replaced by $\phi \Delta q$.

We now define some nomenclature that will be handy in what follows. Let $\hat{n}_x, \hat{n}_y, \hat{n}_z$ be the direction cosines of the cell-face unit normal. Let us define

$$\begin{aligned} n_x &= \hat{n}_x S \\ n_y &= \hat{n}_y S \\ n_z &= \hat{n}_z S \end{aligned} \tag{6.47a}$$

where S is defined in Eq. 6.39, and

$$\begin{aligned} \hat{n}_t &= -(\hat{n}_x \dot{x} + \hat{n}_y \dot{y} + \hat{n}_z \dot{z}) \\ n_t &= -(n_x \dot{x} + n_y \dot{y} + n_z \dot{z}) \end{aligned} \tag{6.47b}$$

We also note here that the eigenvalues and characteristic speeds that arise as a natural part of the various Riemann solvers to be be presented below are useful in computing allowable time steps for explicit time-stepping schemes.

Linear Wave Equation

The linear wave equation in three dimensions was defined in Eq. 5.3. Defining the characteristic speed

$$U = n_t + a n_x + b n_y + c n_z, \tag{6.48}$$

the exact Riemann Solver is given by

$$\begin{aligned} q^R &= q_l \quad \text{if } U > 0 \\ q^R &= q_r \quad \text{if } U \leq 0 \end{aligned} \tag{6.49}$$

Inviscid Burgers' Equation

The inviscid form of Burgers' equation was defined in Eq. 5.4. We first define

$$C = n_t + a n_x + b n_y + c n_z, \tag{6.50}$$

followed by the characteristic velocities

$$\begin{aligned} U_l &= C q_l \\ U_r &= C q_r \quad . \end{aligned} \tag{6.51}$$

Next we note that for the one-dimensional inviscid Burgers' equation

$$\frac{\partial u}{\partial t} + \frac{\partial u^2/2}{\partial x} = 0 \tag{6.52}$$

the characteristic speed is u and the shock speed (assuming $u_l > u_r$) is

$$\dot{x}_s = (u_l + u_r)/2 \quad . \tag{6.53}$$

If $u_l < u_r$ a rarefaction fan is assumed. Extending these ideas to the Riemann problem associated with the cell-face normal direction of the multidimensional problem we can define the exact Riemann Solver as follows.

$$\begin{aligned} &\text{if } U_l \geq U_r, \text{ compute} \\ &U_s = (U_l + U_r)/2 \\ &F^R = U_l q_l/2 \quad \text{if } U_s \geq 0 \\ &F^R = U_r q_r/2 \quad \text{if } U_s < 0 \end{aligned} \tag{6.54}$$

$$\begin{aligned} &\text{if } U_l < U_r, \text{ compute} \\ &F^R = 0 \quad \text{if } U_l < 0 \text{ and } U_r \geq 0 \\ &F^R = U_r q_r/2 \quad \text{if } U_l < 0 \text{ and } U_r < 0 \\ &F^R = U_l q_l/2 \quad \text{if } U_l \geq 0 \end{aligned} \tag{6.55}$$

Euler Equations

For the Euler equations (Eq. 5.5), the three distinct eigenvalues are proportional (by a factor S, see Eq. 6.39) to $\widehat{U} - c, \widehat{U}, \widehat{U} + c$ where

$$\widehat{U} = \hat{n}_t + u\,\hat{n}_x + v\,\hat{n}_y + w\,\hat{n}_z \tag{6.56}$$

The eigenvalue $U = \widehat{U}S = n_t + u\,n_x + v\,n_y + w\,n_z$ is repeated thrice when there are no extra equations for σ (see Eq. 5.5) and is repeated $3 + N$ times when there are N equations for σ. In this report, we only consider the case $N = 0$. and

Godunov Scheme

The Riemann Problem is an initial value problem with piecewise-constant initial data. For the Godunov scheme[14], the exact solution of the Riemann Problem is utilized. The exact solution is made up of constant states separated by transitions in the values of the dependent variables across each family of waves. The wave transitions can be of three types: 1) continuous transition across rarefaction fans, 2) abrupt nonlinear jumps across shock waves, and 3) linearly degenerate jumps across contact surfaces.

Let us consider the entire transition between q_l and q_r (Fig. 6.8). We write the four constant states separating the three wave families as q_0, q_1, q_2, and q_3 where $q_0 = q_l$ and $q_3 = q_r$. If $p_1 > p_0$, the $(\widehat{U} - c)$ wave is a shock. If $p_2 > p_3$, the $\widehat{U} + c$ family is a shock transition. Otherwise, these are rarefaction fans.

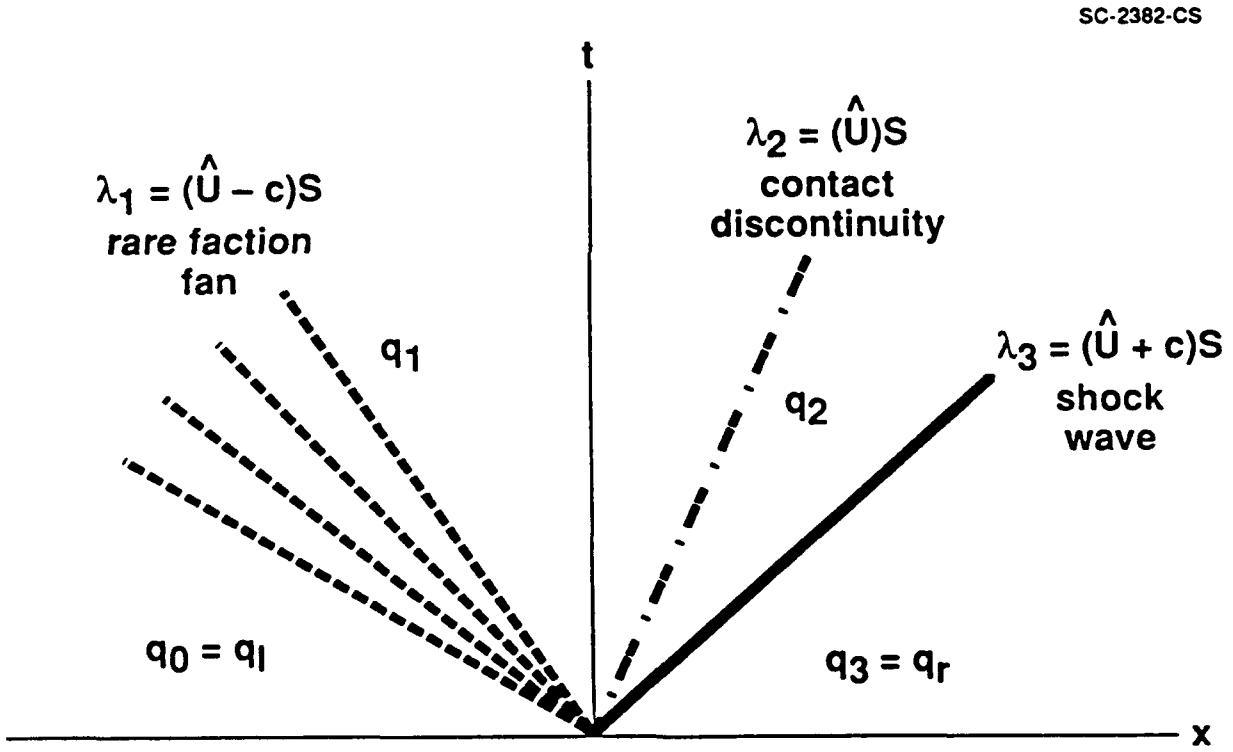


Figure 6.8 Example Riemann problem solution for 1-d Euler equations

The following relationships are valid across the three types of wave transitions:

RAREFACTION — ($\widehat{U} \pm c$) eigenvalues; $i = 1, 3$

$$\begin{aligned} p_{l_i}/\rho_{l_i}^{\gamma} &= p_{r_i}/\rho_{r_i}^{\gamma} \\ \widehat{U}_{l_i} \mp 2c_{l_i}/(\gamma - 1) &= \widehat{U}_{r_i} \mp 2c_{r_i}/(\gamma - 1) \end{aligned} \tag{6.57}$$

SHOCK WAVE — Case 1: $p_{l_i} > p_{r_i}$; ($\widehat{U} + c$) eigenvalue; $i = 3$

$$\begin{aligned} (M_s^2)_{r_i} &= \frac{\gamma + 1}{2\gamma}\left(\frac{p_{l_i}}{p_{r_i}} - 1\right) + 1.0 \\ \frac{\rho_{l_i}}{\rho_{r_i}} &= \frac{(\gamma + 1)}{(\gamma - 1) + 2/(M_s^2)_{r_i}} \\ (\dot{n}_s)_i &= \widehat{U}_{r_i} + c_{r_i}(M_s)_{r_i} \\ \frac{\widehat{U}_{r_i} - (\dot{n}_s)_i}{\widehat{U}_{l_i} - (\dot{n}_s)_i} &= \frac{\rho_{l_i}}{\rho_{r_i}} \end{aligned} \tag{6.58a}$$

SHOCK WAVE — Case 2: $p_{r_i} > p_{l_i}$; ($\widehat{U} - c$) eigenvalue; $i = 1$

$$\begin{aligned} (M_s^2)_{l_i} &= \frac{\gamma + 1}{2\gamma}\left(\frac{p_{r_i}}{p_{l_i}} - 1\right) + 1.0 \\ \frac{\rho_{r_i}}{\rho_{l_i}} &= \frac{(\gamma + 1)}{(\gamma - 1) + 2/(M_s^2)_{l_i}} \\ (\dot{n}_s)_i &= \widehat{U}_r - c_r(M_s)_{l_i} \\ \frac{\widehat{U}_{l_i} - (\dot{n}_s)_i}{\widehat{U}_{r_i} - (\dot{n}_s)_i} &= \frac{\rho_{r_i}}{\rho_{l_i}} \end{aligned} \tag{6.58b}$$

CONTACT DISCONTINUITY; ($\widehat{U}$) eigenvalue; $i = 2$

$$\begin{aligned} p_{l_i} &= p_{r_i} \\ \widehat{U}_{l_i} &= \widehat{U}_{r_i} \end{aligned} \tag{6.59}$$

In the above, we have used subscripts l_i and r_i to mean the constant states to the left and right of the i-th wave transition being considered. For example, when the first

wave is being considered, $l_i = 0$ and $r_i = 1$. The quantity $(M_s)_{l_i}$ is the Mach number based on a) the velocity corresponding to the left state but measured with respect to the moving shock wave, and b) the speed of sound corresponding to the pressure and density of the left state. The quantity $(M_s)_{r_i}$ is the counterpart corresponding to the right state. The associated shock speeds have been denoted by $(\dot{n}_s)_i$.

In the above, the quantities p/ρ^γ and $\widehat{U} \mp \frac{2}{\gamma-1}c$ are Riemann Invariants across the rarefactions identified by the eigenvalues $\widehat{U} \pm c$. Across the contact discontinuity, p and $\widehat{U}$ are Riemann Invariants.

It is clear that there are two equations per wave family — a total of six equations. The unknowns are $(p, \rho, \widehat{U})_1$ and $(p, \rho, \widehat{U})_2$ — also six in number. The six equations are sufficient to evaluate these six unknowns. We must augment these equations with information needed to compute the Cartesian velocity components. The quantity $\widehat{U}$ is the component of velocity along the direction of the normal to the cell face at the quadrature point of interest. The tangential component of velocity remains invariant across rarefaction fans and shock waves. This leads to

$$\begin{aligned} u_1 &= u_0 + (\widehat{U}_1 - \widehat{U}_0)\hat{n}_x \\ v_1 &= v_0 + (\widehat{U}_1 - \widehat{U}_0)\hat{n}_y \\ w_1 &= w_0 + (\widehat{U}_1 - \widehat{U}_0)\hat{n}_z \end{aligned} \tag{6.60}$$

and

$$\begin{aligned} u_2 &= u_3 + (\widehat{U}_2 - \widehat{U}_3)\hat{n}_x \\ v_2 &= v_3 + (\widehat{U}_2 - \widehat{U}_3)\hat{n}_y \\ w_2 &= w_3 + (\widehat{U}_2 - \widehat{U}_3)\hat{n}_z \end{aligned} \tag{6.61}$$

One way of utilizing the exact solution, given above, to the Riemann problem is given below. Given the initial data $q_0 = q_l$ and $q_3 = q_r$, first compute the intermediate states given by q_1 and q_2 in each interval. Define two more intermediate quantities initially to be

$$\begin{aligned} q_{1*} &= q_0 \\ q_{2*} &= q_3 \end{aligned} \tag{6.62}$$

If the $(\widehat{U} - c)$ wave is a rarefaction and $(\widehat{U} - c)_0(\widehat{U} - c)_1 < 0$, then compute q_{1*} (now defined to be a sonic point) from q_0 using Eqs. 6.57 ($l_i = 0, r_i = 1*$) along with the

auxiliary condition

$$\widehat{U}_{1*} - c_{1*} = 0 \tag{6.63a}$$

Similarly, if the $(\widehat{U}+c)$ wave is a rarefaction and the eigenvalue changes sign between 2 and 3, compute the sonic state q_{2*} using Eqs. 6.57 ($l_i = 2*, r_i = 3$) along with the auxiliary sonic condition

$$\widehat{U}_{2*} + c_{2*} = 0 \tag{6.63b}$$

Then, define the various positive and negative flux differences of Eq. 6.45 by

$$\begin{aligned}
dF_1^+ &= \max\left(\text{sign}\{(\widehat{U}-c)_0\},0\right)(F(q_{1*}) - F(q_0)) \\
&\quad + \max\left(\text{sign}\{(\widehat{U}-c)_1\},0\right)(F(q_1) - F(q_{1*})) \\
dF_1^- &= \max\left(-\text{sign}\{(\widehat{U}-c)_0\},0\right)(F(q_{1*}) - F(q_0)) \\
&\quad + \max\left(-\text{sign}\{(\widehat{U}-c)_1\},0\right)(F(q_1) - F(q_{1*})) \\
dF_2^+ &= \max\left(\text{sign}\{\widehat{U}_1\},0\right)(F(q_2) - F(q_1)) \\
dF_2^- &= \max\left(-\text{sign}\{\widehat{U}_1\},0\right)(F(q_2) - F(q_1)) \\
dF_3^+ &= \max\left(\text{sign}\{(\widehat{U}+c)_2\},0\right)(F(q_{2*}) - F(q_2)) \\
&\quad + \max\left(\text{sign}\{(\widehat{U}+c)_3\},0\right)(F(q_3) - F(q_{2*})) \\
dF_3^- &= \max\left(-\text{sign}\{(\widehat{U}+c)_2\},0\right)(F(q_{2*}) - F(q_2)) \\
&\quad + \max\left(-\text{sign}\{(\widehat{U}+c)_3\},0\right)(F(q_3) - F(q_{2*}))
\end{aligned} \tag{6.64}$$

Another approach to utilizing the Godunov scheme is to directly identify q^R and then to compute F^R.

Osher's Scheme

In Godunov's scheme, when any of the waves is a shock, the equations for the intermediate states are not explicitly solvable for the unknowns and iterative techniques must be employed. In contrast, Osher's numerical algorithm uses an approximate solution to the Riemann problem which results in explicit expressions for the intermediate state variables. Osher replaces the shock wave by overturned rarefactions[38].

Thus the wave transition for both nonlinear fields (for the $(\widehat{U}-c)$ field and the $(\widehat{U}+c)$ field) are described in terms of Eqs. 6.57 for rarefaction. The resulting six equations lead to explicit formulae for $\rho_1, \widehat{U}_1, p_1, \rho_2, \widehat{U}_2, p_2$. Then, Eq. 6.60 and Eq. 6.61 can be used to obtain the Cartesian velocity components. Once the intermediate variables are computed, the corresponding fluxes are computed and included in the expressions for $dF_i^{\pm}$ using the same expressions presented earlier for Godunov's scheme. For reasons outlined in an earlier subsection on "Continuous Transitions," the Osher scheme cannot be considered to be a physically correct Riemann Solver in the sense that it assumes overturned rarefactions. For the same reason, we cannot use the Osher scheme to evaluate the "Riemann" flux using Eq. 6.42. Inverted rarefactions notwithstanding, the flux differences that arise in Osher's scheme can be used in Eq. 6.45 to result in a satisfactory numerical algorithm.

Roe's Scheme

Roe's algorithm is based on the exact solution of an approximate Riemann Problem[9]. In Roe's approach, specially averaged cell interface values (denoted by subscript *Roe*) are determined for density, velocity and enthalpy ($h = \gamma p/((\gamma - 1)\rho) + (u^2 + v^2 + w^2)/2$)

$$\begin{aligned}
\rho_{Roe} &= \frac{\rho_r\sqrt{\rho_r} + \rho_l\sqrt{\rho_l}}{\sqrt{\rho_r} + \sqrt{\rho_l}} \\
u_{Roe} &= \frac{u_r\sqrt{\rho_r} + u_l\sqrt{\rho_l}}{\sqrt{\rho_r} + \sqrt{\rho_l}} \\
v_{Roe} &= \frac{v_r\sqrt{\rho_r} + v_l\sqrt{\rho_l}}{\sqrt{\rho_r} + \sqrt{\rho_l}} \\
w_{Roe} &= \frac{w_r\sqrt{\rho_r} + w_l\sqrt{\rho_l}}{\sqrt{\rho_r} + \sqrt{\rho_l}} \\
h_{Roe} &= \frac{h_r\sqrt{\rho_r} + h_l\sqrt{\rho_l}}{\sqrt{\rho_r} + \sqrt{\rho_l}}
\end{aligned} \tag{6.65}$$

from which the speed of sound c can be calculated as

$$c_{Roe} = \sqrt{\{h_{Roe} - (u_{Roe}^2 + v_{Roe}^2 + w_{Roe}^2)/2\}(\gamma - 1)} \quad . \tag{6.66}$$

Using these specially averaged values, Roe evaluates the Jacobian matrix and then considers the approximate, linear, Riemann Problem given by

$$\frac{\partial q}{\partial t} + A_{Roe}\frac{\partial q}{\partial n} = 0 \tag{6.67}$$

at each cell face.

The exact solution to this problem is given for the intermediate states as

$$q_i - q_{i-1} = \alpha_i r_{i\,Roe}, \quad i = 1, \ldots, m \tag{6.68}$$

with the parameters α_i evaluated from the expressions

$$\alpha_i = l_{i\,Roe} \cdot (q_r - q_l), \quad i = 1, \ldots, m \quad . \tag{6.69}$$

In the above equations, l_i are the left eigenvectors of the Jacobian matrix, evaluated such that they are orthonormal to the collection of right eigenvectors r_i. The physical meaning of the parameters α_i can be identified by considering the state space of dependent variables. We have already see that in such a space, the equations for Δq_i (Eq. 6.68) imply that the change Δq_i in dependent variables across each wave family is tangential to the corresponding right eigenvector and that α_i is a measure of the magnitude of that change.

Once again, knowing q_1, q_2, etc., the various fluxes can be evaluated and included in the positive and negative flux differences in the following way which differs from the expressions in Eqs. 6.64 for Godunov's and Osher's schemes only because the Riemann Problem in Roe's scheme is linear, and consequently all the wave transitions are linear jump discontinuities. Additionally, we keep the three repeated eigenvalues U distinct because their corresponding r_i are linearly independent. Therefore $q_0 = q_l$ and $q_5 = q_r$. Thus, the various positive and negative flux differences of Eq. 6.45 are

now defined by

$$
\begin{aligned}
dF_1^+ &= \max[\text{sign}\{(\widehat{U}-c)_{Roe}\},0][f(q_1)-f(q_0)] \\
dF_1^- &= \max[-\text{sign}\{(\widehat{U}-c)_{Roe}\},0][f(q_1)-f(q_0)] \\
dF_2^+ &= \max[\text{sign}\{\widehat{U}_{Roe}\},0][f(q_2)-f(q_1)] \\
dF_2^- &= \max[-\text{sign}\{\widehat{U}_{Roe}\},0][f(q_2)-f(q_1)] \\
dF_3^+ &= \max[\text{sign}\{\widehat{U}_{Roe}\},0][f(q_3)-f(q_2)] \\
dF_3^- &= \max[-\text{sign}\{\widehat{U}_{Roe}\},0][f(q_3)-f(q_2)] \\
dF_4^+ &= \max[\text{sign}\{\widehat{U}_{Roe}\},0][f(q_4)-f(q_3)] \\
dF_4^- &= \max[-\text{sign}\{\widehat{U}_{Roe}\},0][f(q_4)-f(q_3)] \\
dF_5^+ &= \max[\text{sign}\{(\widehat{U}+c)_{Roe}\},0][f(q_5)-f(q_4)] \\
dF_5^- &= \max[-\text{sign}\{(\widehat{U}+c)_{Roe}\},0][f(q_5)-f(q_4)]
\end{aligned}
\tag{6.70}
$$

For Roe's scheme, the values of $dF_i^{\pm}$ can also be directly defined to be

$$
dF_i^{\pm} = \frac{(\lambda_{iRoe} \pm |\lambda_{iRoe}|)}{2} \alpha_i r_{iRoe} \quad , \quad i = 1,\ldots,m \quad . \tag{6.71}
$$

We can therefore also write

$$
|\Delta F| = (R|\Lambda|L)_{Roe}(q_r - q_l) \quad . \tag{6.72}
$$

where R is the matrix of right eigenvectors, L is the matrix of left eigenvectors (with $RL = LR = I$) and $|\Lambda|$ is the diagonal matrix of absolute values of the eigenvalues $\lambda_i, i = 1,\ldots,m$.

Because Roe's scheme converts the actual Riemann problem to an approximate one that is linear, there are no continuous transitions. This approach does not rule out expansion shocks. In order to avoid expansion shocks, a modification to the fluxes is implemented at sonic rarefactions.

$$
\begin{aligned}
&\textit{Replace } dF_i^{\pm} \textit{ by } \pm \frac{[\lambda_i(q_r) - \lambda_i(q_l)]}{4} \alpha_i r_{iRoe} \\
&\textit{iff } \lambda_i(q_l) < 0 < \lambda_i(q_r)
\end{aligned}
\tag{6.73}
$$

It must be noted that the augmented diffusion is only added to the actual field requiring it and not to all wave families at a sonic rarefaction.

Harten-Lax Scheme

The Harten-Lax scheme will be described in future editions of this report.

Rusanov Scheme

The Rusanov scheme can be applied within the present framework if we begin with Eq. 6.72 and replace $|\Lambda|$ with its spectral radius $|\lambda|_{max}$

$$|\Delta F| = |\lambda|_{max} (q_r - q_l) \quad . \tag{6.74}$$

Lax-Friedrichs Scheme

The Lax-Friedrichs scheme can be applied within the present framework if we begin with Eq. 6.74 and replace the spectral radius with a positive constant ϵ

$$|\Delta F| = \epsilon (q_r - q_l) \quad . \tag{6.75}$$

7.0 GEOMETRY FORMULATION DETAILS

The geometry formulation in UNIVERSE-series unstructured grid codes is very flexible and, in general, can handle any type of element (conservation cell). In particular, we have included three types of cells at present. These are 1) the hexahedral cell, 2) the triangular prism cell, and 3) the tetrahedral cell. The details of the geometry treatment for these three cell types are given in this section.

Local Node Numbers and Coordinates

A cell type is defined by the number and placement of its vertices. A complete definition of a cell type would aiso include, in addition to the vertices, any other nodes needed to define the variation of the geometric variables x, y, z in the cell (see Fig. 4.2 on higher-order elements). Taken together, the vertex nodes and other nodes can simply be referred to as the nodes of the cell.

We assume a local coordinate system for each cell (ξ, η, σ, Fig. 4.1). The vertex node numbers are defined in terms of an order set of integers and the values of the local coordinates at each vertex are defined for each cell type in Figs. 7.1, 7.2 and 7.3.

Shape Function

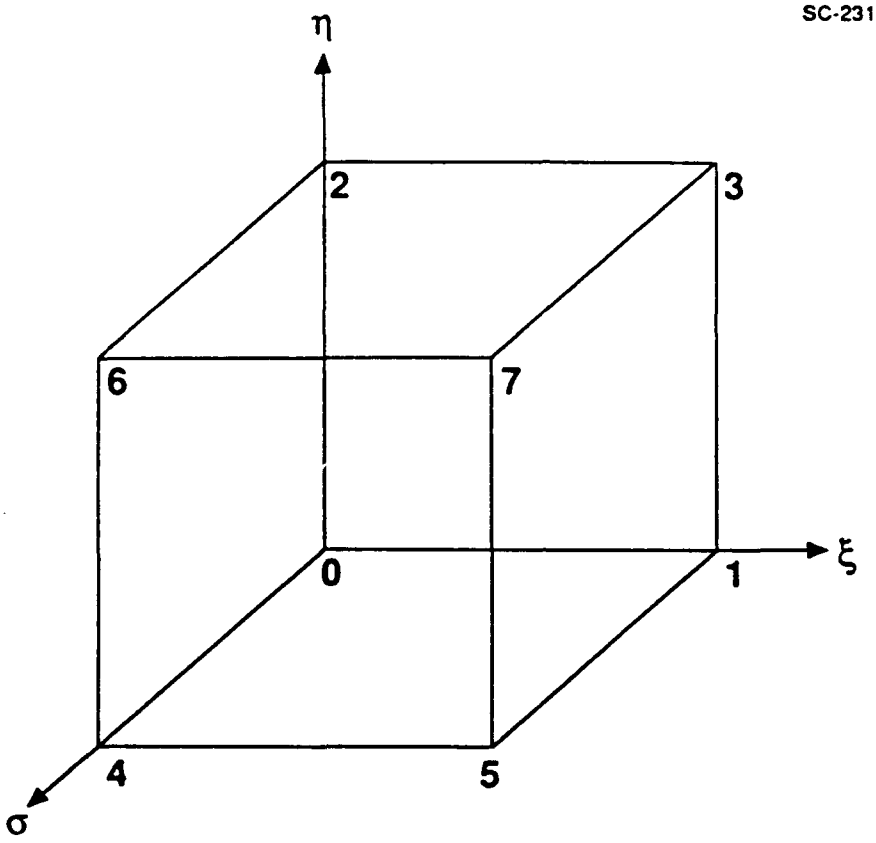
The geometric "shape function" $N_i(\xi, \eta, \sigma)$ was defined in Eqs. 4.9-11 for the three cell types. Each cell node is associated with its own shape function which is a multidimensional polynomial in ξ, η, σ:

$$N_i(\xi, \eta, \sigma) = \sum_{k=0}^{K} p_{ik} \xi^{l(k)} \eta^{m(k)} \sigma^{n(k)} \tag{7.1}$$

The polynomial coefficients can easily be obtained by solving the K equations resulting from setting

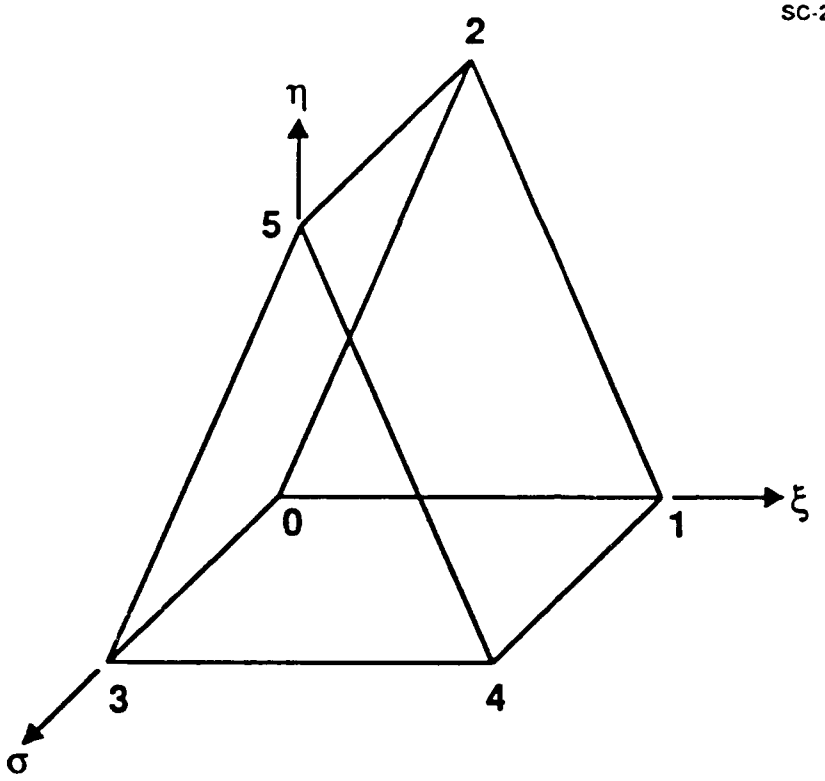
$$N_i(\xi_i, \eta_i, \sigma_i) = 1, \tag{7.2a}$$

$$N_i(\xi_j, \eta_j, \sigma_j) = 0, \quad j \neq i. \tag{7.2a}$$



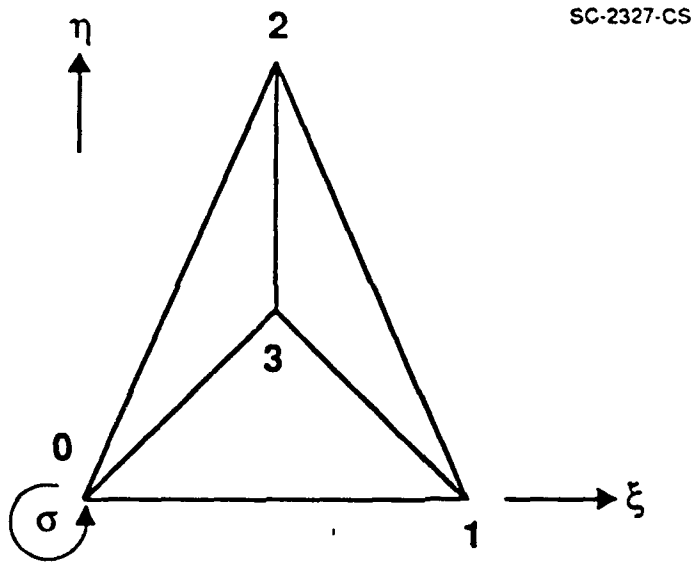
VERTEX	ξ	η	σ
0	−1	−1	−1
1	+1	−1	−1
2	−1	+1	−1
3	+1	+1	−1
4	−1	−1	+1
5	+1	−1	+1
6	−1	+1	+1
7	+1	+1	+1

Figure 7.1 Vertex nodes and coordinates for hexahedron



Vertex	ξ	η	σ
0	-1	-1	-1
1	+1	-1	-1
2	0	+1	-1
3	-1	-1	+1
4	+1	-1	+1
5	0	+1	+1

Figure 7.2 Vertex nodes and coordinates for triangular prism



Vertex	ξ	η	σ
0	-1	-1	-1
1	+1	-1	-1
2	0	1	-1
3	0	0	+1

Figure 7.3 Vertex nodes and coordinates for tetrahedron

When the three-dimensional geometry polynomial is built by forming the product of two or more lower-dimensional polynomials (Eq. 4.10, Eq. 4.11), the three-dimensional shape functions can also be constructed from the appropriate lower-dimensional shape functions.

Local Faces and Vertices

Each face of a cell is defined by which vertices belong to it (in terms of local node numbers). The faces and their corresponding vertices are identified for each of the three cell types in Figs. 7.4, 7.5 and 7.6.

Cell-Face Metrics and Normals

The method used to compute cell-face normals was outlined in Section 4 in the paragraphs following Eq. 4.11. A more detailed description is given below.

Given the geometry polynomials (Eq. 4.9, Eq. 4.10, or Eq. 4.11), the tangents to the local coordinate directions can be defined by

$$\begin{aligned} \vec{\xi} &= x_\xi \hat{j} + y_\xi \hat{k} + z_\xi \hat{l} \\ \vec{\eta} &= x_\eta \hat{j} + y_\eta \hat{k} + z_\eta \hat{l} \\ \vec{\sigma} &= x_\sigma \hat{j} + y_\sigma \hat{k} + z_\sigma \hat{l} \end{aligned} \tag{7.3}$$

Next we identify two vectors in the plane of the cell face by choosing suitable linear combinations of the above.

$$\begin{pmatrix} \vec{V}_1 \\ \vec{V}_2 \end{pmatrix} = \begin{pmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \end{pmatrix} \begin{pmatrix} \vec{\xi} \\ \vec{\eta} \\ \vec{\sigma} \end{pmatrix} \tag{7.4}$$

The transformation matrix in the above can be defined as T

$$T = \begin{pmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \end{pmatrix} \tag{7.5}$$

and is defined for each face of each cell type in Figs. 7.7, 7.8 and 7.9. The T matrix elements are selected appropriately to account for the fact that ξ, η, σ range, in each

cell, from -1 to $+1$ in many cases. The cross product of $\vec{V}_1$ and $\vec{V}_2$ defines the cell-face normal.

Cell-Face Quadrature Points

At every cell face, the integral of any quantity Q over that face can be replaced, upto the desired order of accuracy, by a suitable quadrature formula

$$\int_t \int_s Q(s,t)\, ds\, dt = \sum_{\text{quads.}} C_i\, Q(s_i, t_i) \tag{7.6}$$

where s, t are the representative running orthogonal coordinates tangential to the face, and subscript i denotes a quadrature point. The coefficient C_i is the "weight" assigned to the i-th quadrature point. For the three cell types being considered, one encounters only two types of cell faces — "square" or "triangular" (in terms of the running coordinates s and t, as well as the local coordinates ξ, η, σ). Tables 7.1 and 7.2 represent four-point Gaussian quadrature points and weights for such "square" and "triangular" cells. This leads to fourth-order accuracy in the evaluation of cell-face integrals. Figure 7.10 presents these quadrature points diagrammatically (not necessarily to true scale).

s	t	C
−0.577350269189626	−0.577350269189626	0.5
+0.577350269189626	−0.577350269189626	0.5
−0.577350269189626	+0.577350269189626	0.5
+0.577350269189626	+0.577350269189626	0.5

Table 7.1 Gaussian quadrature points for "square" face

s	t	C
−0.487831473351	−0.689897894859	0.318041443825
+0.487831473351	−0.689897894859	0.318041443825
−0.204988807440	+0.289898127317	0.181958556175
+0.204988807440	+0.289898127317	0.181958556175

Table 7.2 Gaussian quadrature points for "triangular" face

Special Cases

While, in general, the four-point Gaussian quadrature is employed for three-dimensional problems, simpler formulae (with fewer quadrature points) can be used sometimes without any loss of accuracy. For example, with hexahedral cells, the midpoint rule is sufficient for one-dimensional problems in the one significant direction being considered. For two-dimensional problems, a two-point quadrature is sufficient for faces spanning the third direction (the two points are along the line that bifurcates the cell in the third direction).

Calculation of Cell Volume

The cell volume is related to Eqs. 4.18 and 4.19

$$V = \int\int\int_V \vec{\nabla} \cdot (x\hat{j} + y\hat{k} + z\hat{l})\, dV \tag{7.7}$$

corresponding to $i = 0$ in Eq. 4.18. The volume integration can be replaced by surface integration.

$$\begin{aligned} V &= \int\int\int_V (\vec{\nabla} \cdot \vec{X}_0)\, dV \\ &= \int\int_S (\vec{X}_0 \cdot \hat{n})\, dS \end{aligned} \tag{7.8}$$

The surface integral can, in turn, be converted to a quadrature formula based on Eq. 7.6.

Strong-Conservation-Law Form in General Coordinates

In Section 5, the strong-conservation-law form was introduced for hyperbolic systems of conservation laws in Cartesian coordinates. In future editions of this report, the invariance of that form under coordinate transformations will be discussed.

Integral and Strong-Conservation-Law Forms

There is a clear and useful geometric analogy between the integral form and the strong-conservation-law form of the equations. This will be elucidated in future editions of this report.

Preservation of Uniform Flow

Physically, uniform flow remains uniform. Numerically, this simple fact may not be true. In other words, when a numerical method is applied to update the solution from time step to time step with uniform flow as the starting solution, on nonuniform grids only a careful construction of the algorithm will guarantee that the numerical solution does not introduce spurious disturbances into uniform flow. These issues will be considered in detail in future editions of this report.

SC-2318-CS

Face	Vertex 0	Vertex 1	Vertex 2	Vertex 3
0	0	1	3	2
1	4	5	7	6
2	0	1	5	4
3	2	3	7	6
4	0	2	6	4
5	1	3	7	5

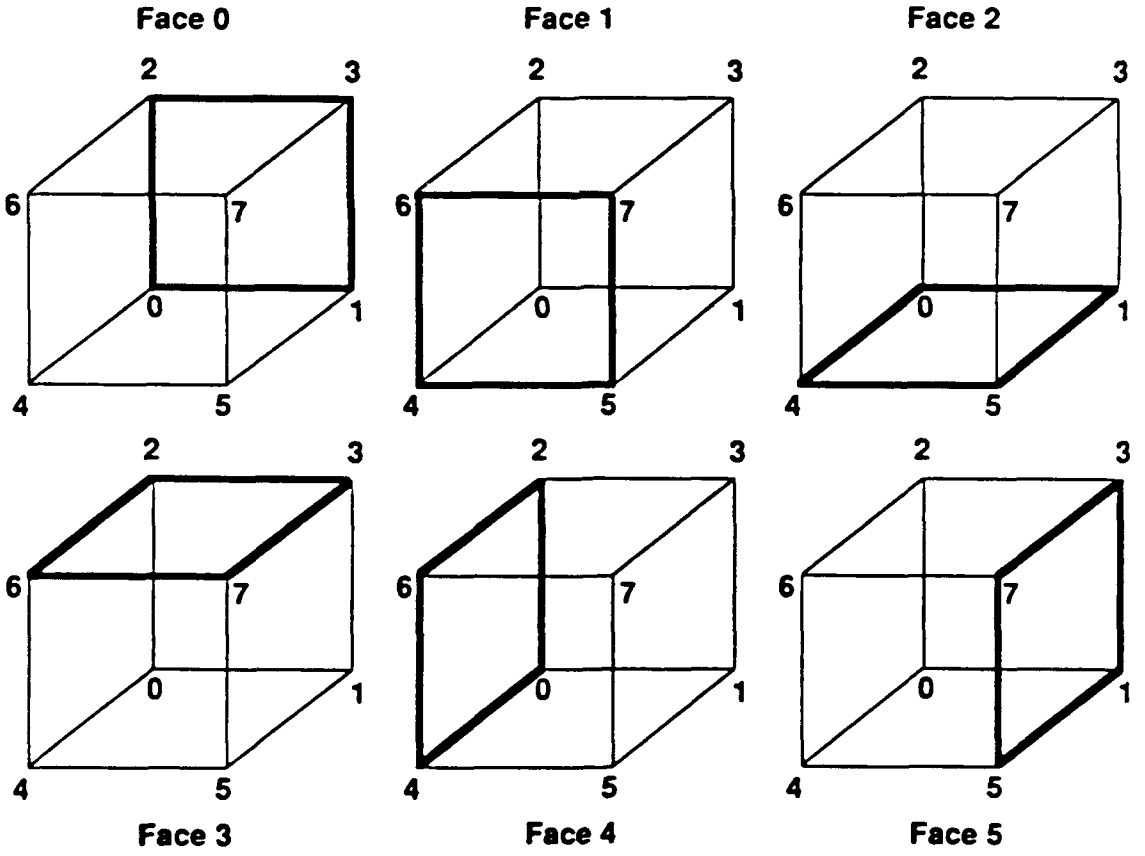


Figure 7.4 Faces and their vertices for hexahedron

SC-2323-CS

Face	Vertex 0	Vertex 1	Vertex 2	Vertex 3
0	0	1	2	–
1	3	4	5	–
2	0	1	4	3
3	0	2	5	3
4	1	2	5	4

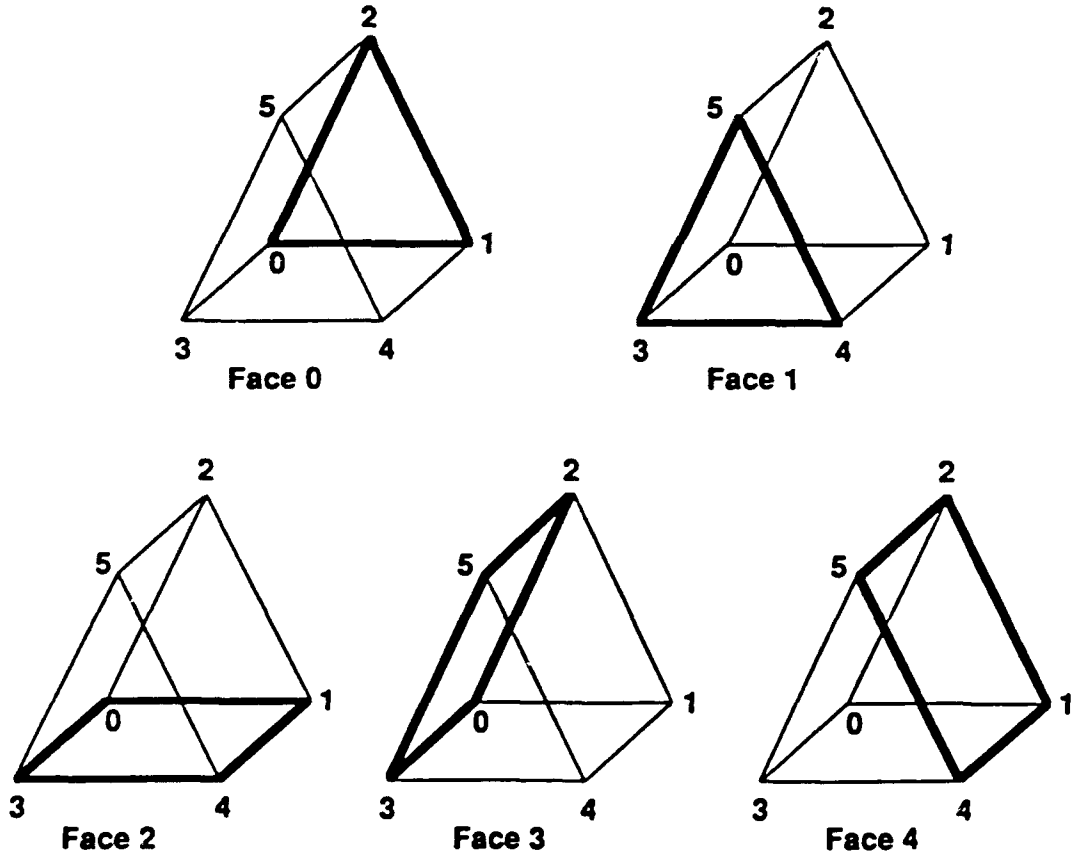


Figure 7.5 Faces and their vertices for triangular prism

SC-2328-CS

Face	Vertex 0	Vertex 1	Vertex 2
0	0	1	2
1	0	1	3
2	1	2	3
3	0	2	3

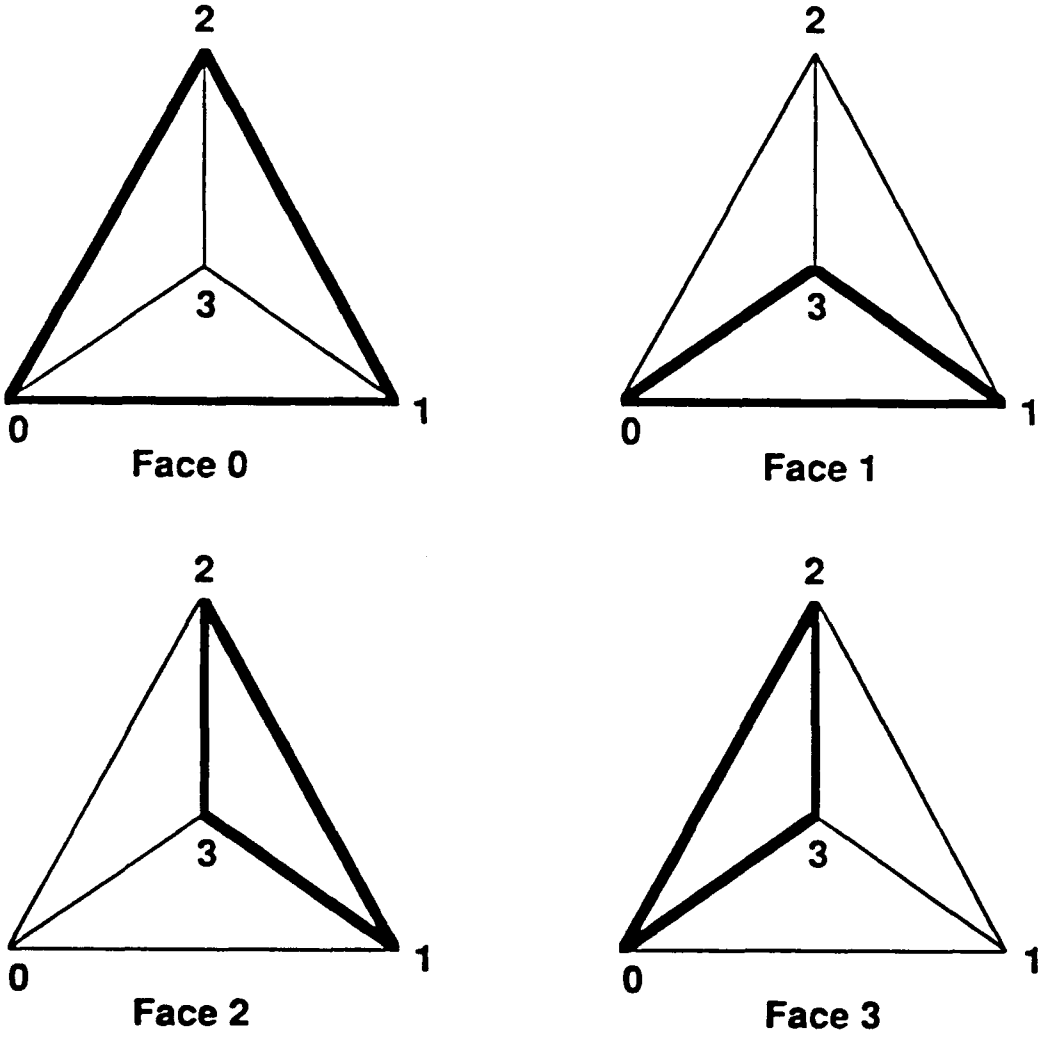


Figure 7.6 Faces and their vertices for tetrahedron

SC-2319-CS

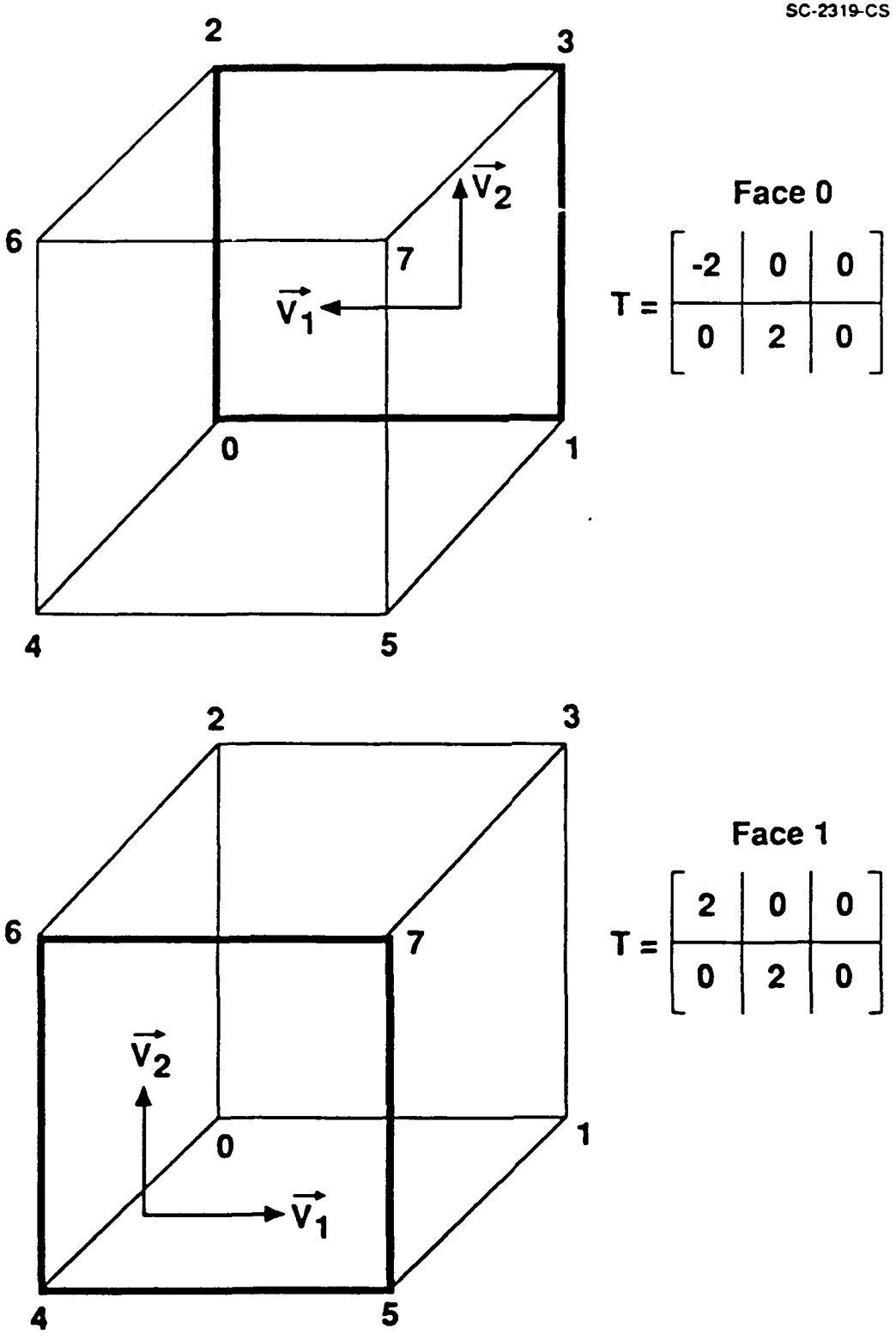
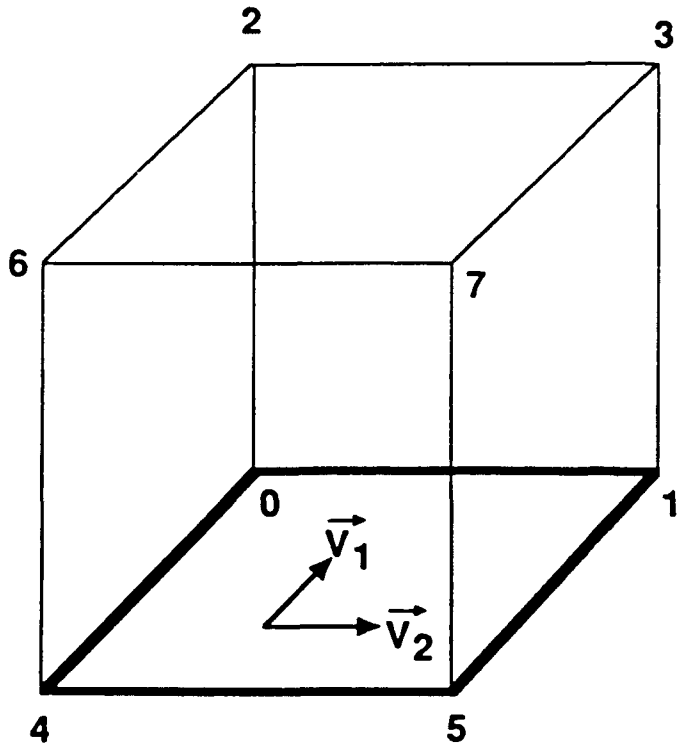


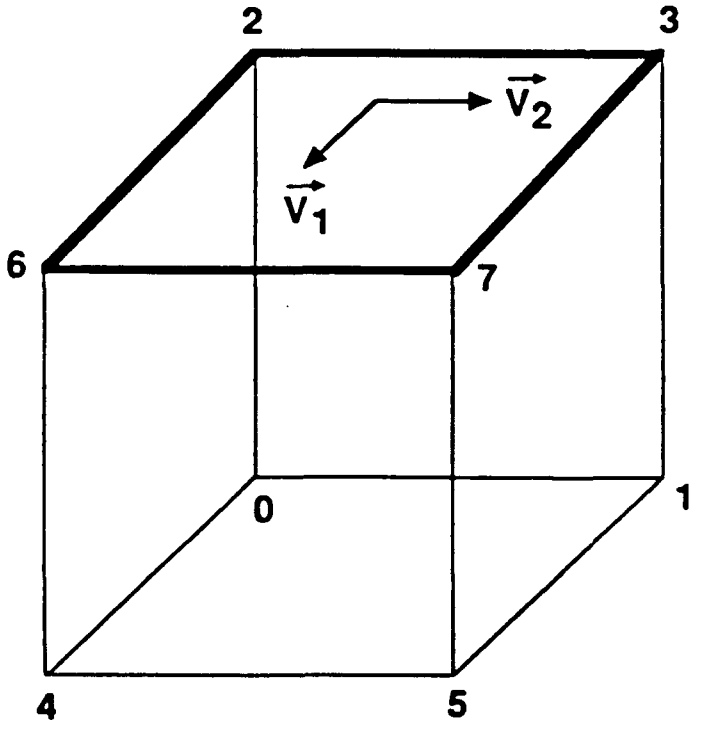
Figure 7.7 Faces and their transformation matrix for hexahedron

SC-2320-CS



Face 2

$$T = \begin{bmatrix} 0 & 0 & -2 \\ 2 & 0 & 0 \end{bmatrix}$$

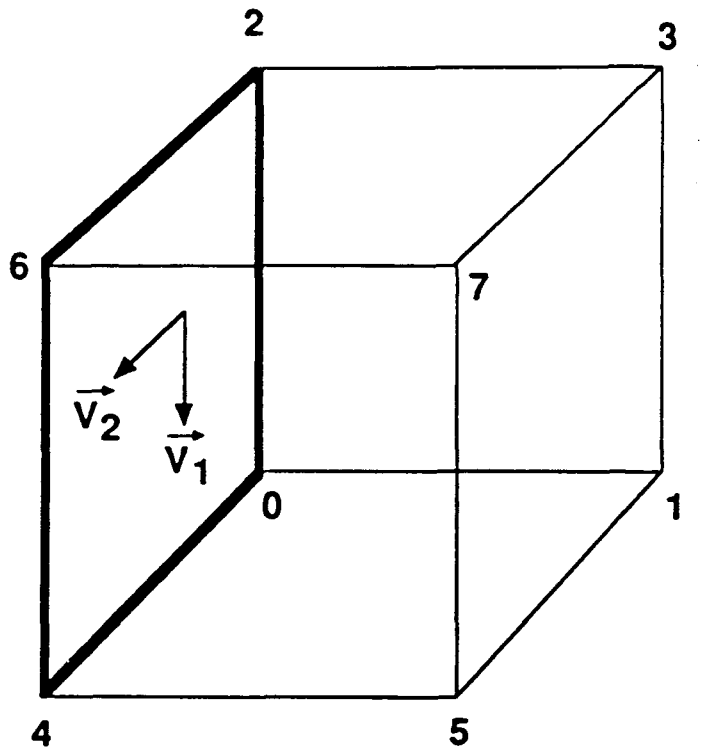


Face 3

$$T = \begin{bmatrix} 0 & 0 & 2 \\ 2 & 0 & 0 \end{bmatrix}$$

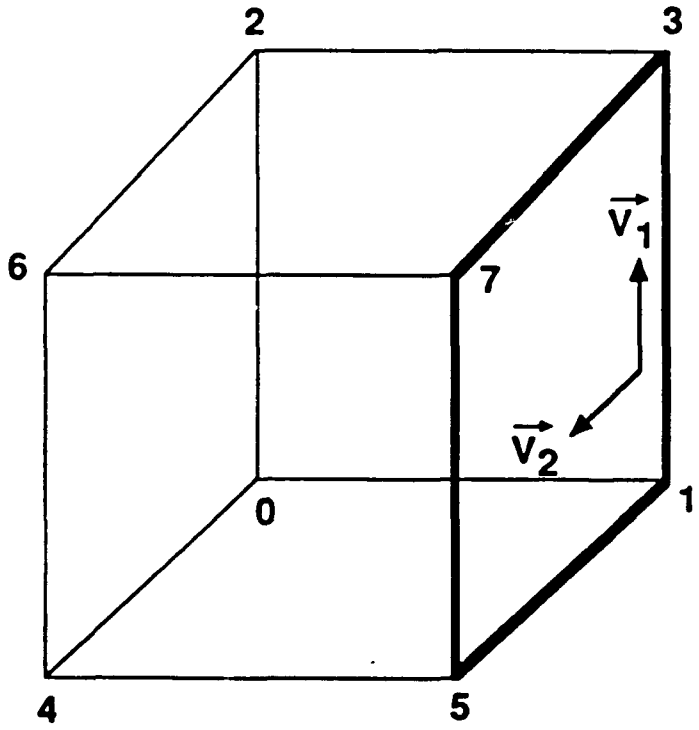
Figure 7.7 Faces and their transformation matrix for hexahedron — continued

SC-2321-CS



Face 4

$$T = \begin{bmatrix} 0 & -2 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$



Face 5

$$T = \begin{bmatrix} 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

Figure 7.7 Faces and their transformation matrix for hexahedron — continued

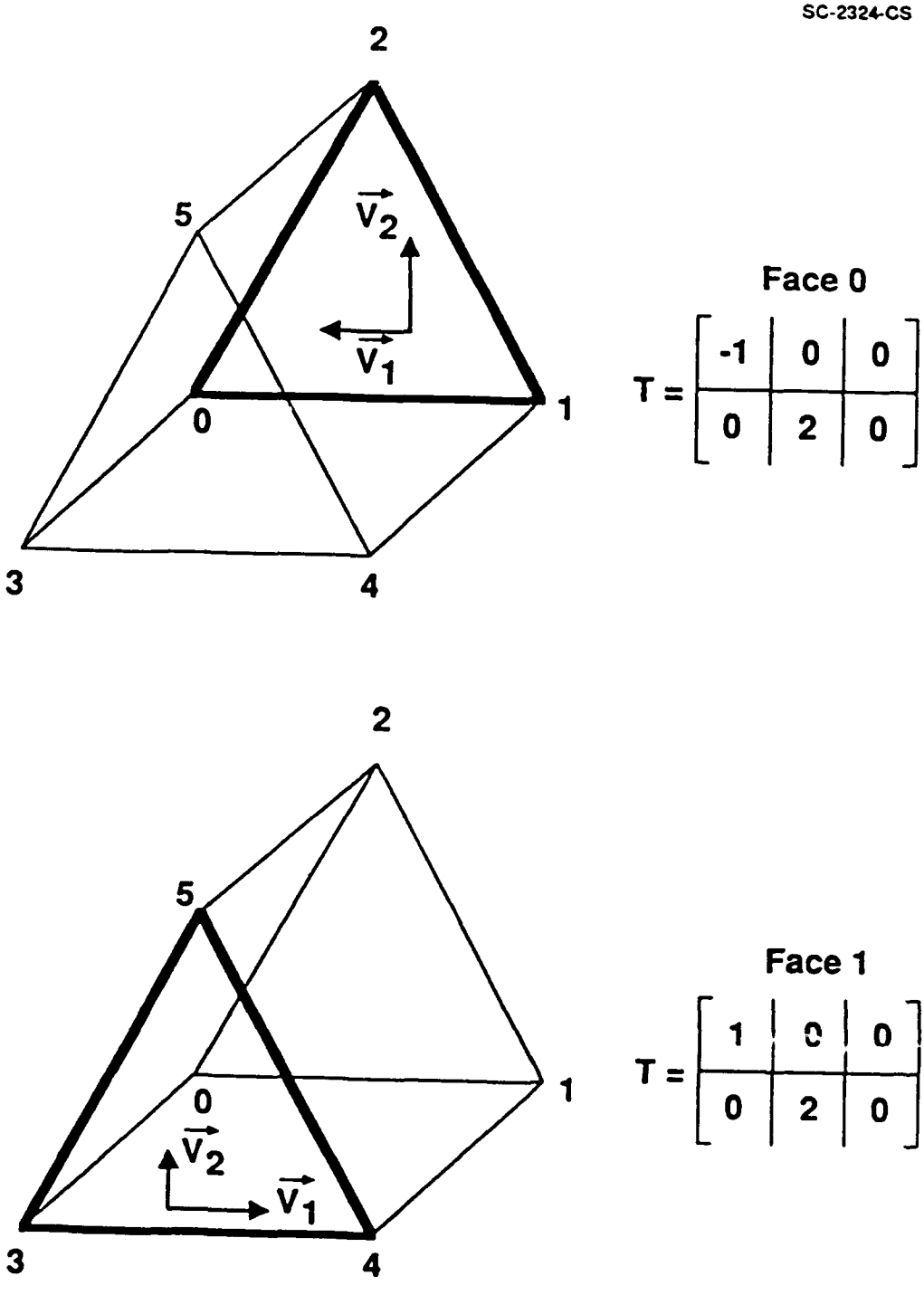
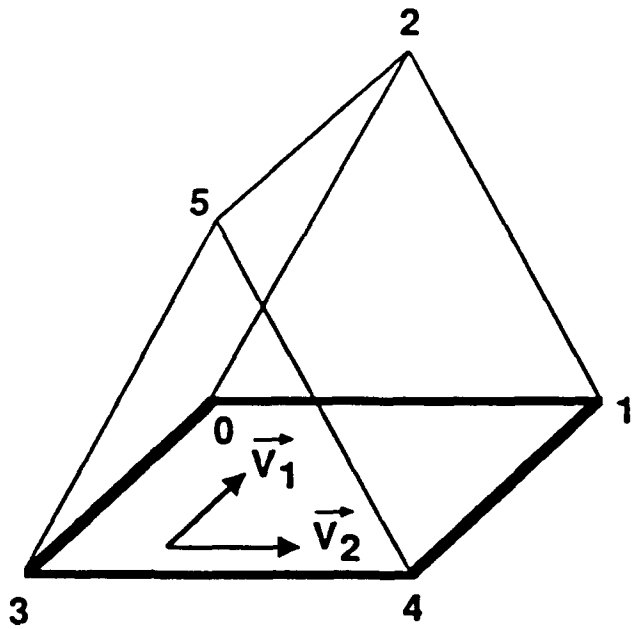


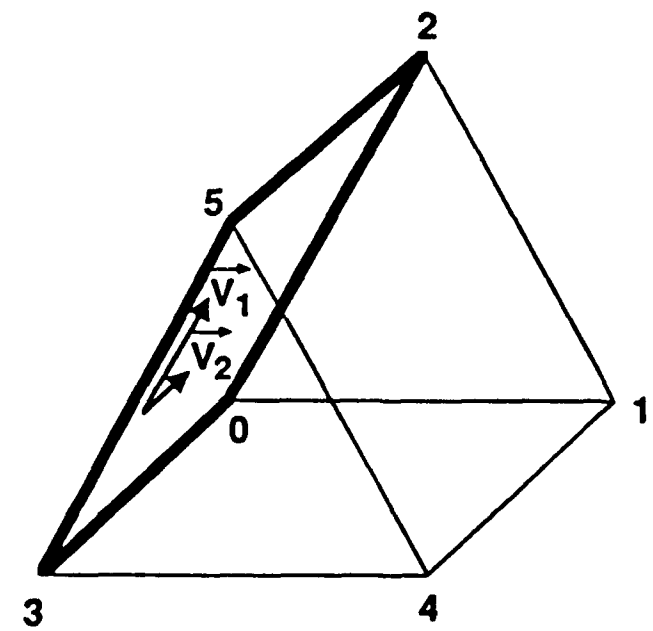
Figure 7.8 Faces and their transformation matrix for triangular prism

SC-2325-CS



Face 2

$$T = \left[\begin{array}{c|c|c} 0 & 0 & -2 \\ \hline 2 & 0 & 0 \end{array}\right]$$



Face 3

$$T = \left[\begin{array}{c|c|c} 1 & 2 & 0 \\ \hline 0 & 0 & -2 \end{array}\right]$$

Figure 7.8 Faces and their transformation matrix for triangular prism — continued

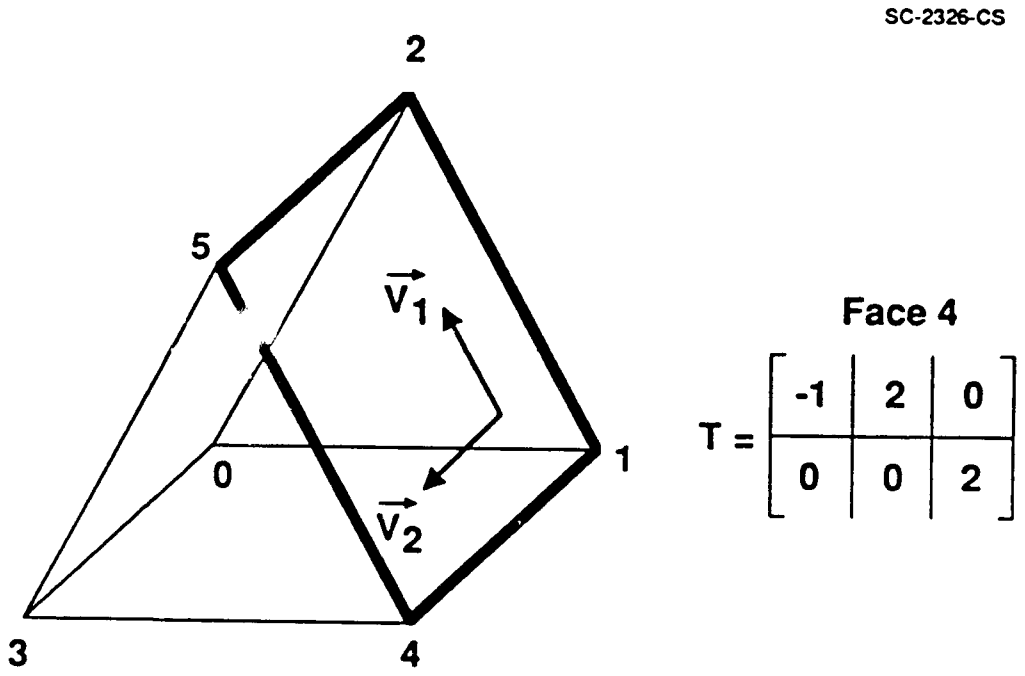
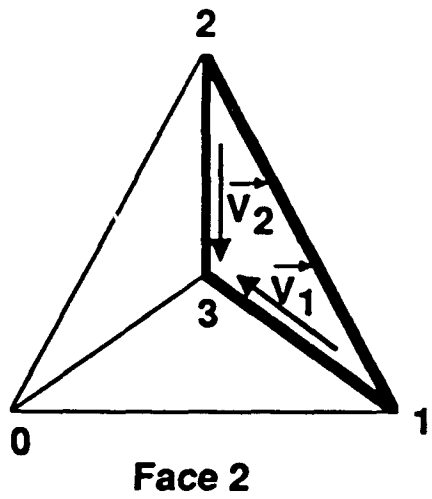


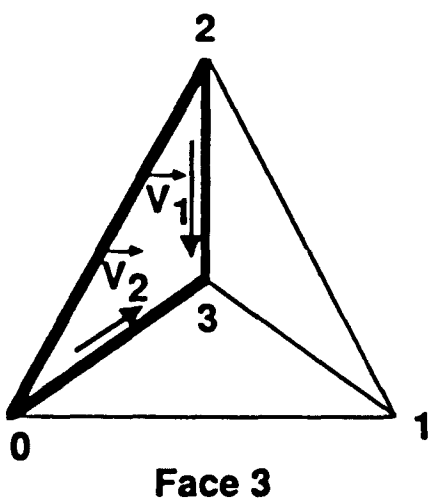
Figure 7.8 Faces and their transformation matrix for triangular prism — continued

SC-2330-CS



Face 2

$$T = \left[\begin{array}{c|c|c} -\frac{1}{2} & \frac{1}{2} & 1 \\ \hline 0 & -1 & 2 \end{array}\right]$$

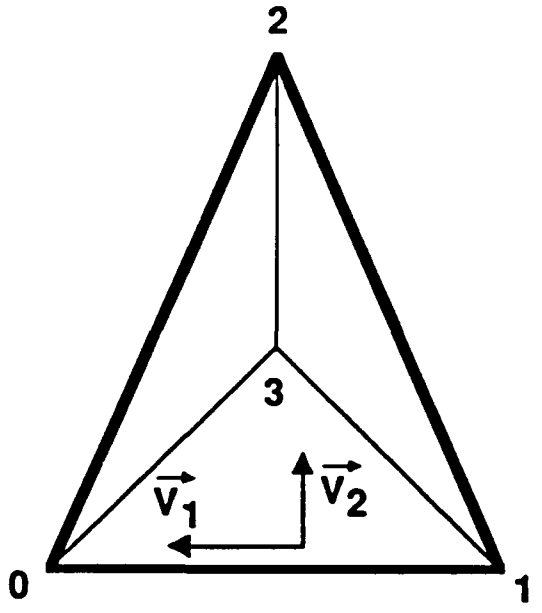


Face 3

$$T = \left[\begin{array}{c|c|c} 0 & -\frac{1}{2} & 1 \\ \hline 1 & 1 & 2 \end{array}\right]$$

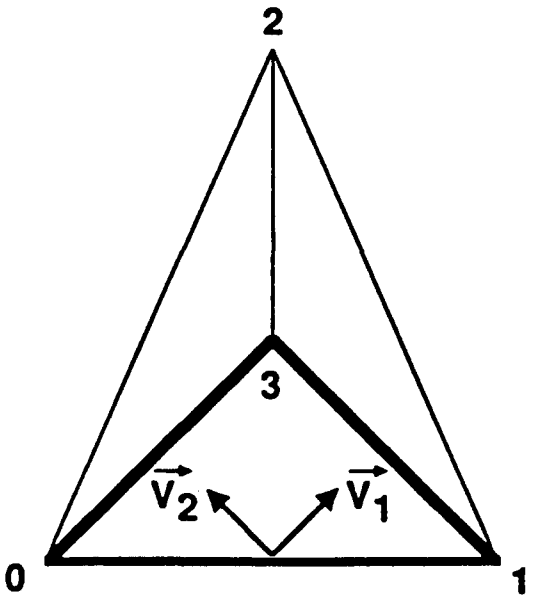
Figure 7.9 Faces and their transformation matrix for tetrahedron

SC-2329-CS



Face 0

$$T = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix}$$



Face 1

$$T = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 1 \\ -1 & 1 & 2 \end{bmatrix}$$

Figure 7.9 Faces and their transformation matrix for tetrahedron — continued

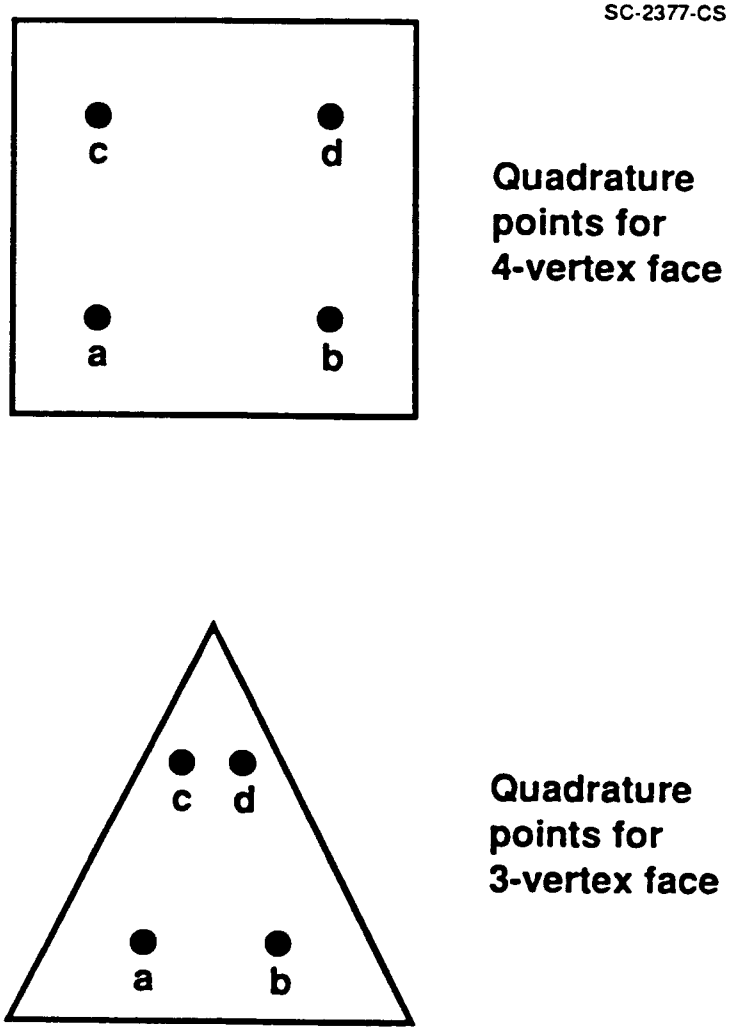


Figure 7.10 Gaussian quadrature points for "square" and "triangular" faces

8.0 TOPOLOGY TREATMENT

This section describes various topology issues related to connectivity of nodes into cells, cells common to a node, cells with a common face, how cells may be subdivided, how links may be removed, etc.

Node Locations and Nodes of Cell

The computational unstructured mesh is defined by 1) specifying the set of nodes and their locations (x, y, z coordinates and optionally their nodal velocities $\dot{x}, \dot{y}, \dot{z}$), and 2) specifying the set of nodes that comprise each cell. These node numbers are called the **global** node numbers. Each cell has an ordered collection of nodes numbered locally. When the global nodes of each cell are provided, they must be given in the same order as the local node numbers. Therefore, given any local node number, the corresponding global node number is easily determined.

Elimination of Redundant Nodes and Cells

Let us make a set of all nodes that are part of atleast one cell in the total collection of cells given. This may be a subset of the collection of nodes for which the locations have been provided. The extra nodes in the larger set are those nodes not needed to define any cell. These nodes (and their locations) may have been generated during the mesh generation process but were not used because their inclusion would have resulted in cells that were somehow unsatisfactory. The ability to remove redundant nodes is provided for in the unstructured-grid UNIVERSE-series formulation.

During the process of removing links, a topic to be covered later in this section, some cells may become degenerate. This may happen in two ways: 1) the number of distinct global vertex node numbers is less than four; 2) the number of distinct vertex node locations is less than four. Case 1 deals with a "logically" degenerate cell and case 2 with a "physically" degenerate cell. Removing these cells become redundant for the purpose of computing the dependent variables. Removing such redundant cells corresponding to case 1 is available in UNIVERSE-series unstructured-grid formulations as part of various "grid editing" options.

Cells of Given Node

Given the nodes of each cell, the inverse map can be constructed. This provides knowledge of which cells include the given node. While each cell has the same number of nodes (assuming the same cell type for every cell), the number of cells that include a given node varies with each node. We have verified for linear elements (only vertex nodes) that the sum of cells associated with each node is equal to the number of cells times the number of nodes per cell which is a somewhat surprizing statement at first glance.

Common Faces

Each face is made up of a specific set of local node numbers, each of which is mapped on to a global node number. Using the knowledge of nodes of a cell and cells of a node, the cells that share a common face can easily be identified.

Faces are numbered sequentially with cell number. The first six faces belong to the first cell, the next six faces belong to the second cell, etc., for the hexahedral cell type. Therefore, face numbers are related directly to cell numbers. A face shared by two cells sports two face numbers, one that identifies it as part of the first cell and another that identifies it as part of the second.

A given face of a given cell has a number of vertex nodes. Each vertex node is associated with many cells. Out of these, one cell is the cell pointed to by the face number being considered. There is not more than one more cell which is common to all nodes of the face. In this fashion, the cells that share a given face can be determined. It then becomes a relatively trivial matter to identify which face of the second cell is identical to the face being considered. Thus the face number corresponding to the neighbor cell can be identified, given the face number of a particular cell. Of course, at a boundary, a face can only be part of one cell, the interior cell.

Cell Division

For various reasons, a cell may have to be divided into two cells. In Phase I, cell division capability has been developed for the triangular prism cell type for use in

two-dimensional inviscid store-tracking studies. Figure 8.1 displays a situation where one face of a triangular prism cell has been divided. If the face is common to two cells, this leads to both cells being subdivided into two cells each.

The following book-keeping details must be kept track of:

1. New nodes are introduced (with global numbers greater than the existing maximum.
2. New cells must be constructed (with cell numbers assigned to be greater than the existing maximum values).
3. Two existing cells must be redefined to account for their being made of different nodes after subdivision.
4. "Cells of node" must be recomputed in the vicinity of the subdivided cells.
5. "Face to face" correspondence must also be reestablished locally.

Such a grid "editing" capability has been developed.

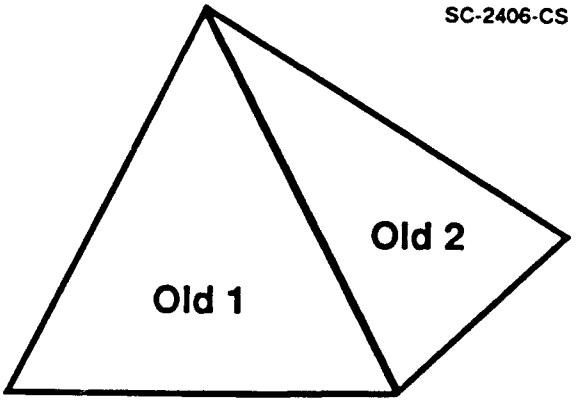
Link Removal

It is sometimes desirable to remove links. In Fig. 8.2, the short link ab is an example. The link can be removed by moving one of its nodes towards the other. In this example, cells 2 and 5 become degenerate and must be removed from consideration. Node b merges with a and therefore a is used to replace b in all cell definitions. Consequently, node b becomes redundant and can be removed from consideration if necessary. The book-keeping steps are given below.

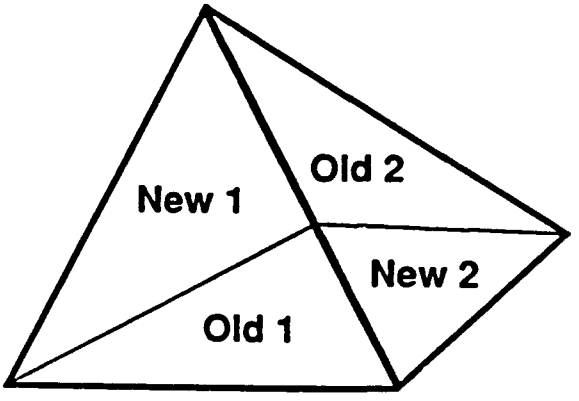
1. Node b for all cells associated with b must be replaced with node a (nodes of cells 3 and 4 must be redefined).
2. Collapsed cells 2 and 5 must be removed from the database.
3. If node b is not removed from database and must be used for some purpose, the location coordinates of b must be modified to coincide with node a's position.

4. "Cells of node" and "face of face" must be recomputed in the local region.

   Such a grid editing capability has also been developed.



Original cell



After cell division

Figure 8.1 Cell division for triangular prisms

SC-2407-CS

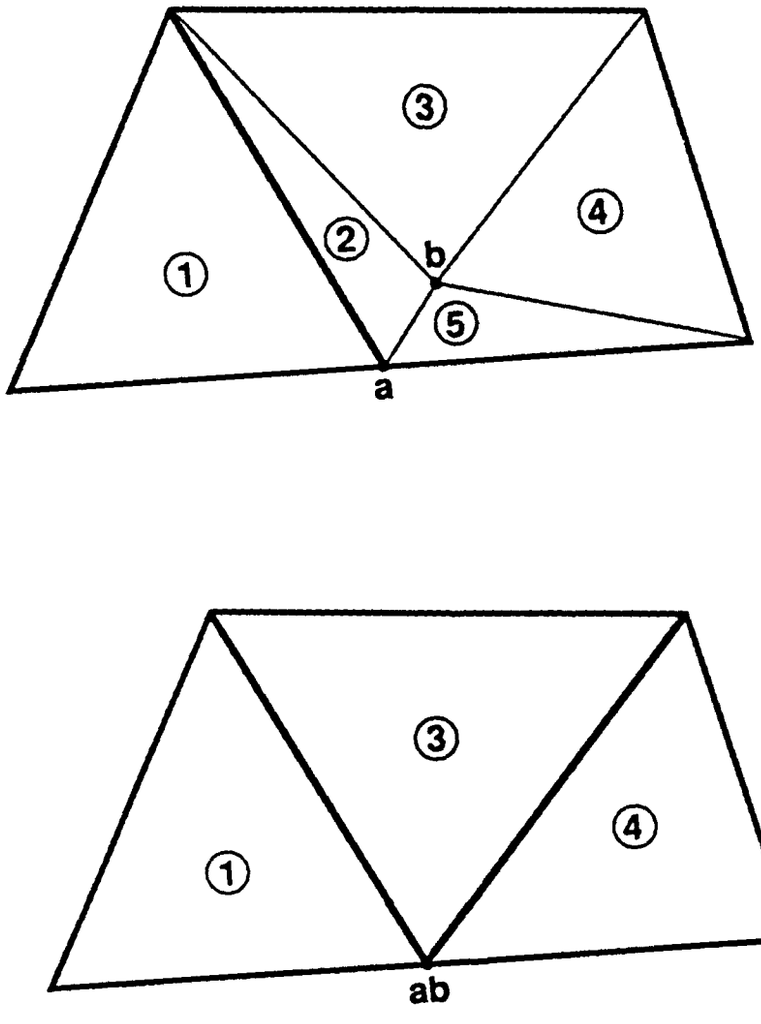


Figure 8.2 Link removal for triangular prisms

9.0 GRID GENERATION METHODOLOGY

An unstructured 2-d mesh generation capability based on a distance function concept[39] had been developed prior to the commencement of the effort described in this report. While this approach was satisfactory based on its generality and versatility, it required substantially more computer time than another method developed during Phase I. Therefore, here we do not make any further reference to the above grid generation methodology. However the two grid generators, TRIM2D and TRIM3D, developed during Phase I will continue to be used during Phase II and are described here.

TRIM2D

TRIM2D is a two-dimensional triangular element mesh generator that was delivered to NWC during Phase I. The underlying algorithm is based on a two-dimensional triangulation procedure that is based on constructing triangles from a cloud of points. The cloud of points can be provided by existing structured grid generators or from a random point generator. For example, the cloud of grid points that was used to generate the unstructured grid shown in Fig. 9.1 was created separately for the fuselage and for the bomb using a structured grid generator. The two sets of points may overlap. A deletion capability has been implemented for those points occurring right at the same position or too close each other. Similarly, redundant points inside the fuselage and bomb are also deleted.

Triangulation is accomplished with the Delaunay triangulation using an advancing front technique. The initial front is composed of all the boundary edges. The detail of the process is described below.

1. An edge of the front is arbitrarily picked up as a candidate for triangulation. The neighboring points on the advancing side of present edge and the adjacent two edges are first collected and sorted by the distance from the present edge.

2. A circle through an advancing-side node and two edge endpoints is constructed. If there is any point enclosed in such a circle, this node is replaced by the enclosed point. This process is repeated until no points can be found in a circumcircle,

the final point is picked for triangulation and the present edge is deleted from the front list. If there is any new edge generated, they should be included in the front list. Figure 9.2 serves to illustrate this approach.

3. The procedure (1)-(2) continues until no front edge exists. The final grid should be similar to that obtained using any other method based on Delaunay triangulation. This is because the Delaunay triangulation procedure uniquely makes triangles out of a given set of points in such a way that the circumcircle of any triangle contains no other points.

TRIM3D

A new three-dimensional unstructured grid generation method, related to the above 2-d technique, has also been developed. Chronologically, TRIM3D was developed by Dr. Kuo-Yen Szema before TRIM2D was developed by Dr. Chung-Lung Chen, both of Rockwell Science Center. A brief description of this technique is presented below for tetrahedral cells.

The boundary of the computational domain is divided into several patches. The geometry of each patch is described by specifying a sufficient number of non-intersecting lines on the patch. Each line, in turn, is described in terms of a sufficient number of points on the line (Fig. 9.3). These lines are arbitrary and are specified by the user. From this information a parametric representation (s, t) of the surface is developed where s and t are the local running coordinates in the plane of the surface. For convenience, the parameters s and t are chosen to take values between 0 and 1, where $s = 0$, $s = 1$, $t = 0$ and $t = 1$ form the boundary of the patch. Nodes are then distributed on these four boundary lines satisfying the user-specified clustering requirements. Presently, the code requires that the number of nodes on at least any one pair of opposite sides ($s = 0$ and $s = 1$ or $t = 0$ and $t = 1$) be equal. This restriction will be removed in the future.

Internal lines are generated by connecting corresponding points on a pair of opposite sides which have the same number of nodes. Each internal line is then divided into line segments. The length of line segments at the ends of an internal line is determined by the average of the length of the line segments on the boundary of the patch that intersect the given internal line. The length of the remaining line

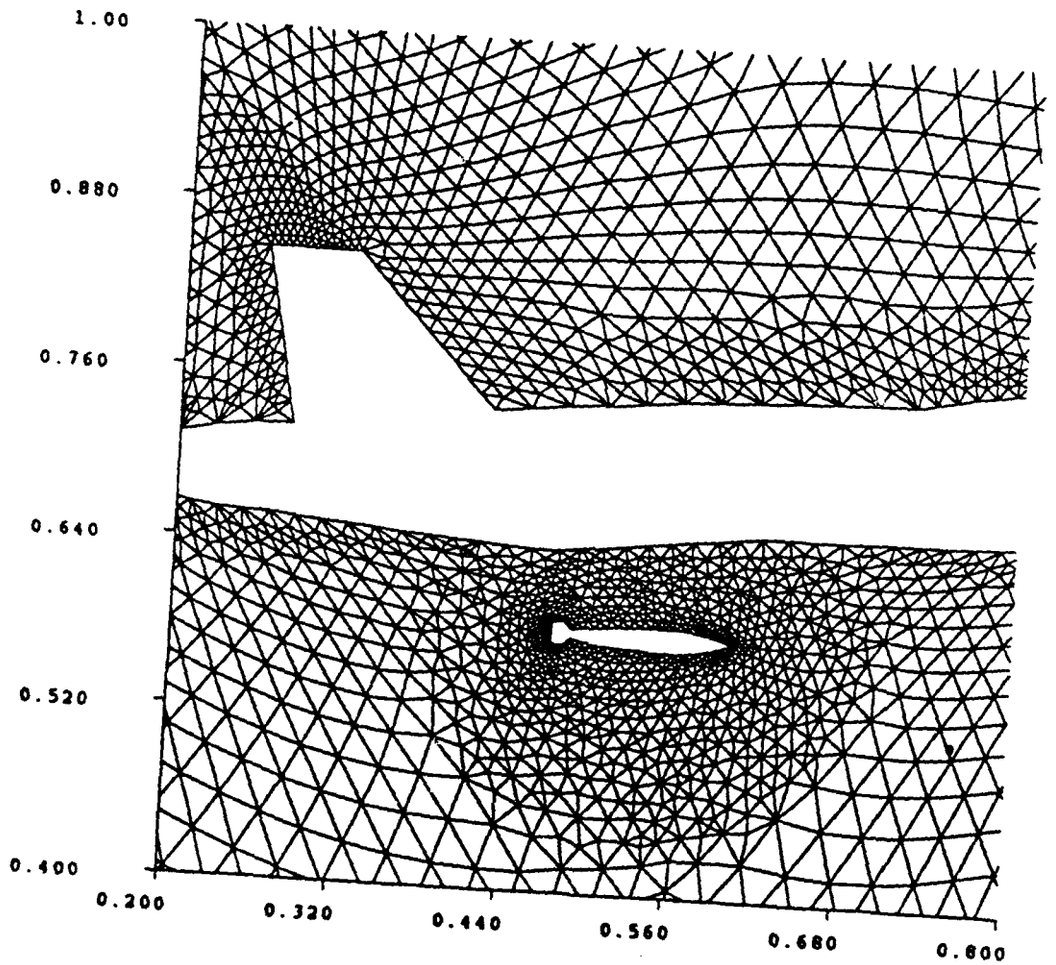


Fig. 9.1 2-d triangular mesh with fuselage and store

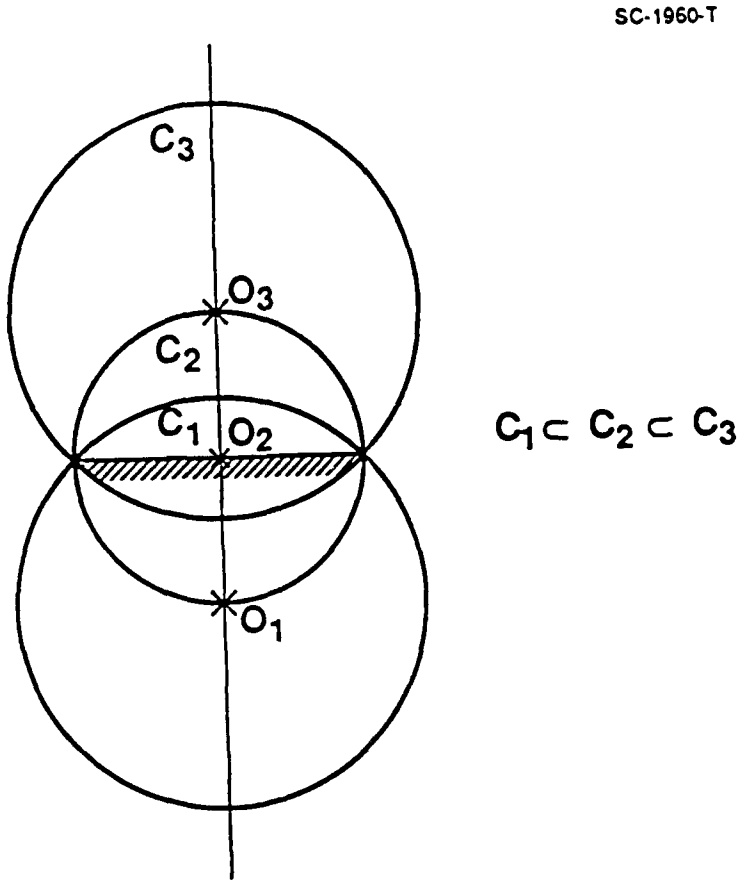


Fig. 9.2 Testing advancing front/Delauney triangulation

segments on an internal line is obtained by smoothly blending the lengths of line segments at the ends. Triangular elements are then generated by connecting the line segments according to the recipe described in Fig. 9.4.

The process of starting from triangular elements on the boundary of the computational domain and filling up the domain with tetrahedra employs the method of "advancing front" in Ref. 40. This technique requires specification of an initial surface divided into triangular elements. The boundary of the computational domain is used as the intial surface of the advancing front. The smallest element on the surface is replaced by the three sides of a new tetrahedron constructed with that element as the base. The fourth node of the tetrahedron is obtained either from one of the existing nodes or by creating a new one (Fig. 9.5).

The TRIM3D grid generation consists of the following steps:

(1) Define the boundary patches (including body surface and outer boundary).

(2) Triangular elements are then generated on each patch based on user supplied information regarding the number of points and mesh streching.

(3) These triangular elements are used as initial surface of the advancing front.

(4) Tetrahedral cells are then contructed with the triangular elements as bases. In order to avoid large cells intersecting with small cells, the smallest triangular element from the list of faces is always used.

(5) Select or create a point to form a tetrahedral cell from the base triangular element (ABC, in Fig. 9.6).

   a.) The point D can belong to a neighboring triangular element (ACD) as long as it can generate a tetrahedron satisfying certain built-in constraints. See Fig. 9.6a (top figure).

   b.) The point D can be selected from an existing grid or from a list of points supplied for the purpose. See Fig. 9.6b (middle figure).

   c.) Otherwise, a new point D can be created to form the new tetrahedron. See Fig. 9.6c (lower figure).

(6) Check whether the newly formed cell intersects with any already existing cells.

(7) If no suitable point can be found, some old cells have to be removed,and repeat step (5).

(8) The new faces ABD and BCD from (5-a) or ABD, BCD, CAD from (5-b) and (5-c), the new cell ABCD and new point D from (5-c) are all added to their respective lists (face, cell and point).

(9) Delete the old face (ABC) from the list.

(10) Check if there are any triangular elements remaining in the front . If yes, go back to step 4.

The two most important aspects of TRIM3D are 1) data structure and 2) the logic to check whether any two tetrahedra intersect with each other. These aspects are disscussed in the following section.

Data Structure

In order to find the smallest triangular element, points in the neighborhood and neighboring triangular elements efficiently, the data structure for the faces, and the grid points is arranged as follows:

List for The Front Face

A binary tree structure is used to form the front face list. The ordering of the tree is arranged such that the area of the father face is smaller than the face of the two sons. Figure 9.7 shows a binary tree of this form. The area of each face and its position are also shown in this figure. It is noted that the position of the two sons are located at $Ison1 = Ifather \times 2$ and $Ison2 = Ifather \times 2 + 1$, respectively. For example, Face D located in position 4 has two sons located at position 8 (H) and 9 (I). A face should be added or deleted without altering the binary tree ordering. The ideas of the heap-sort and heap-search algorithms (Ref. 41) are used in TRIM3D.

a) Adding a new face to the face list:

A new face is always added first at the end of the tree. Then, the internal order of the binary tree is rearranged by comparing the face areas of fathers and their sons. This procedure can be described as follows:

1) A face J with area 4.0 is to be added to a binary tree in Fig. 9.8.

2) Place J at the end of heap list (10 in this case).

3) Find the father's position of 10 by using $Ifather = Ison/2$. Therefore the father's position is 5 and the face is E in this case.

4) Compare the area of faces J and E.

5) If the area of J is less than that of face E, interchange the position of father and son (see Fig. 9.8a).

6) Unless J is at position 1 (top of the list) go back to (3).

b) Delete a face from the front face list:

A face can be removed from any position from the tree. Then, the internal order of the tree is re-established by comparing the area of the face of the father and son.

1) Find the position from which the face is to be removed from the tree. For example, we can consider face A at position 1 in Fig. 9.8b.

2) Place the face stored at the end of the tree at this position (1 for this case).

3) Find the two sons location by using $Ison1 = Ifather \times 2$ and $Ison2 = Ifather \times 2 + 1$.

4) Determine if the father and son's position should be changed by checking the area of the father face and two sons' faces.

if

$(Area(father) < min(Area(son1), Area(son2))$: no change

else

change father's position with the smaller area son's position.

5) Repeat step 4 until no change is reqired (Fig. 9.9).

In this binary tree, the face with smallest area will always remain at the top of the list and can be used as the next surface to be removed.

List for Points in the Neighborhood

An octree data structure is used to efficiently locate points in a neighborhood. The fist octant is determined from the boundary of the computational domain. At most, eight points are stored in each octant. If ninth point falls into an octant, then it subdivided into eight smaller octants. This procedure is continued until an octant with vacant storage is found. Figure 9.10 illustrates this process. A new point I is added and it falls into octant 1 which already contains eight points A,B,C,D,E,F,G,H. It has then to be divided into eight smaller octants (octants 2-9). The newly added point I with old points D,E,F are relocated in octant 2. In the TRIM3D code ,an array $IOCTR(11, MXOCT)$ is defined to store the points. The variable $MXOCT$ is the maximum number of octants allowed. For each octant IOC, the following information is stored.

$IOCTR(11, IOC) = -1$: the octant is full

$IOCTR(11, IOC) = 0$: the octant is empty

$IOCTR(11, IOC) > 0$: the number of points stored in this octant

$IOCTR(1:8, IOC)$: point numbers are stored here if $IOCTR(11, IOC) > 0$

The maximum and minimum x, y, z are also stored for each octant. By using this octree, the neighboring points within some specific distance of a given point (x, y, z) can easily be determined.

Linked List Between Point and Faces

In an unstructured grid generation code, it is important to determine the faces that surround a given point. In order to do that, an address pointer array "$LPION(IPONT)$"
for each grid point IG is first allocated. The faces surrounding point "IG" are saved

in an array $LFAPO(8, IG)$. The procedure to find faces surrounding a given point IG is described below:

1) Find the address related to the given point IG.

$$IADRES = LPION(IG)$$

2) $LFAPO(8, IADRES) > 0$ defines number of stored faces.
$LFAPO(8, IADRES) < 0$ implies that next face number surrounding the grid point IG is continued at the address $abs(LFAPO(8, IADRES))$.

3) $LFAPO(1:7, IADRES) = 0$ defines an empty location
$LFAPO(1:7, IADRES) > 0$ denotes face number for face which surrounds the point IG. This method is illustrated in Fig. 9.11.

To Check for Intersecting Faces

In the process of generating cells, no two faces must intersect each other. This condition can be satisfied only if no side of either face intersects the other face (Figure 9.12). A total of six conditions have to be checked to make sure these two faces do not cross each other. With the notation given in the figure, determine whether a side $\vec{g}_3$ (connecting $\vec{x}_4$ and $\vec{x}_5$) intersects the face $\vec{g}_1\vec{g}_2$ $(\vec{x}_1, \vec{x}_2, \vec{x}_3)$. The intersection point of line $(\vec{x}_4\vec{x}_5)$ with the plane defined by $\vec{x}_1$, $\vec{x}_2$ and $\vec{x}_3$ is given by

$$\vec{x}_I = \vec{x}_4 + \frac{(\vec{g}_1 \times \vec{g}_2) \cdot (\vec{x}_1 - \vec{x}_4)}{(\vec{g}_1 \times \vec{g}_2) \cdot \vec{g}_3} \vec{g}_3$$

The segment $(\vec{x}_4\vec{x}_5)$ intersects the triangle $(\vec{x}_1, \vec{x}_2, \vec{x}_3)$ if $\vec{x}_I$ belongs to the triangle itself. This procedure leads to three conditions for each triangle. If all six conditions are satisfied, then the two faces do not cross.

Mesh Smoothing

The capability to "smooth" mesh point locations without changing cell connectivities has been developed. Four methods are now compared in order to show their

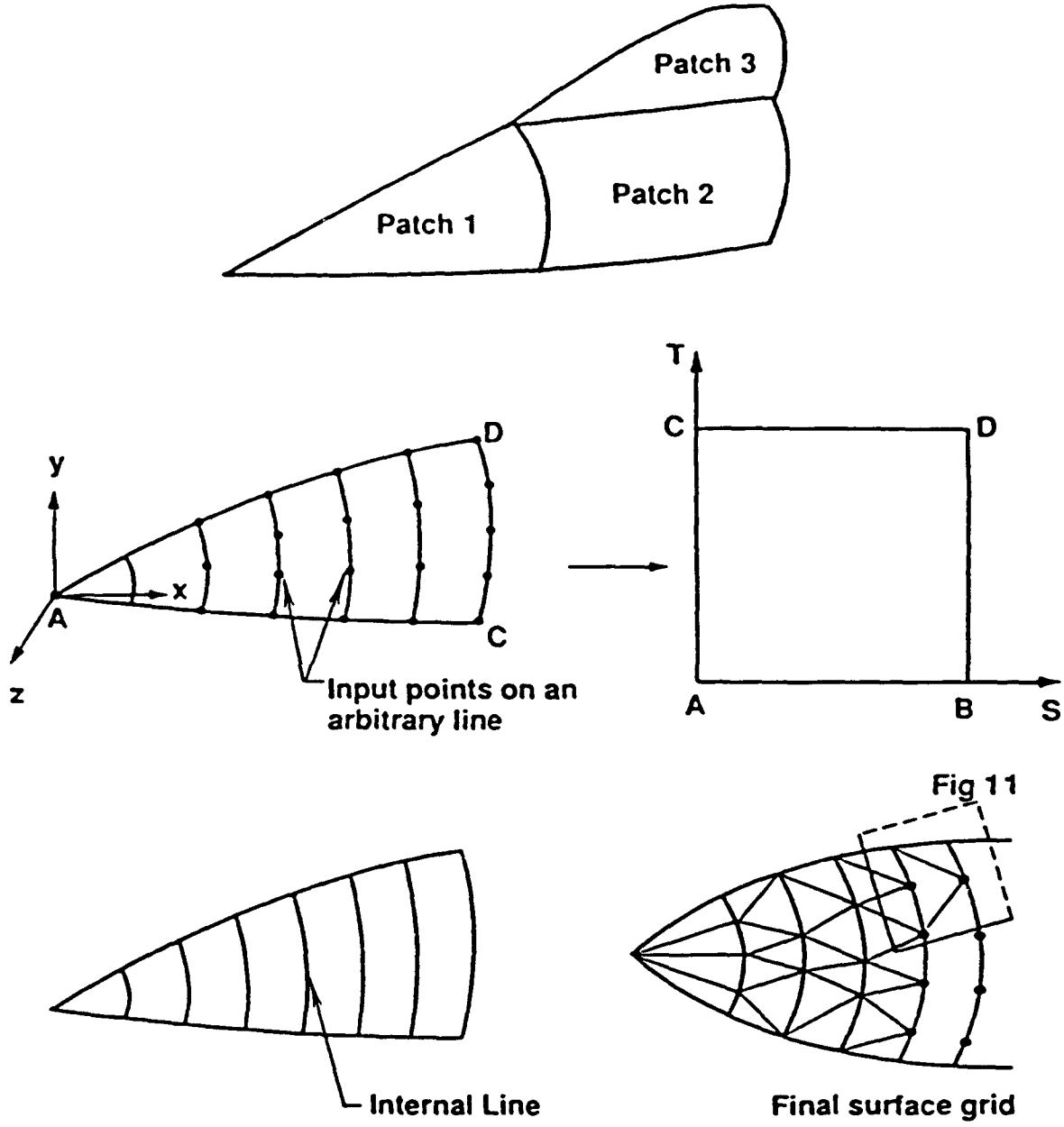
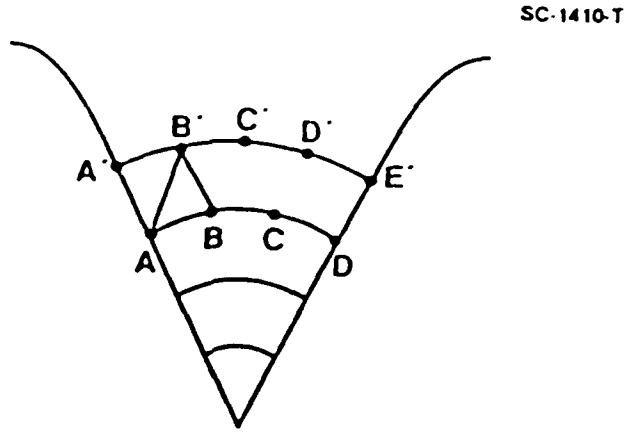
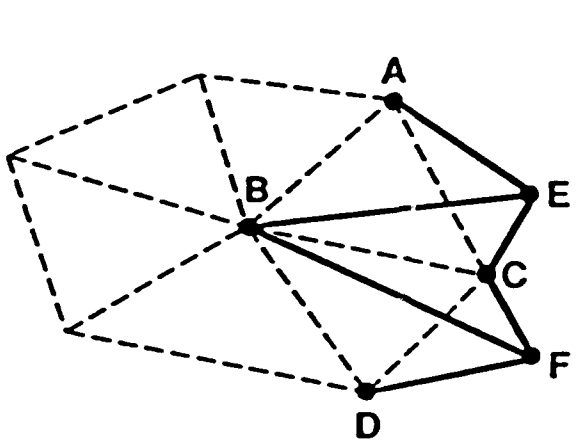


Fig. 9.3 Surface grid generation



1) start with (A,B).
2) There are 2 choices:
   a) A can be connected to B'
   b) B can be connected to A'
3) compute lengths of AB' and BA' and choose the shorter one; for e.g., AB'.
4) since B is not connected yet, continue the process with (A,B).
5) 2 choices :
   a) A can be connected to C'
   b) B can be connected to B'
6) compute lengths of AC' and BB' and choose the shorter one; for e.g., BB'.
7) since both A and B are now connected, continue the process with (B,C).

Fig. 9.4 Recipe for generating triangular elements



- - - - : initial front surface divided into tringular elements.

ABC & CBD : two of the initial surface elements.

E & F : newly generated nodes.

ABE , BCE , CAE : new elements that replace ABC.

CBF , BDF , DCF : new elements that replace CBD.

BFE & FCE : new elements generated by connecting two existing nodes E and F. They replace the elements BCE and CBF.

Fig. 9.5 Advancing front technique

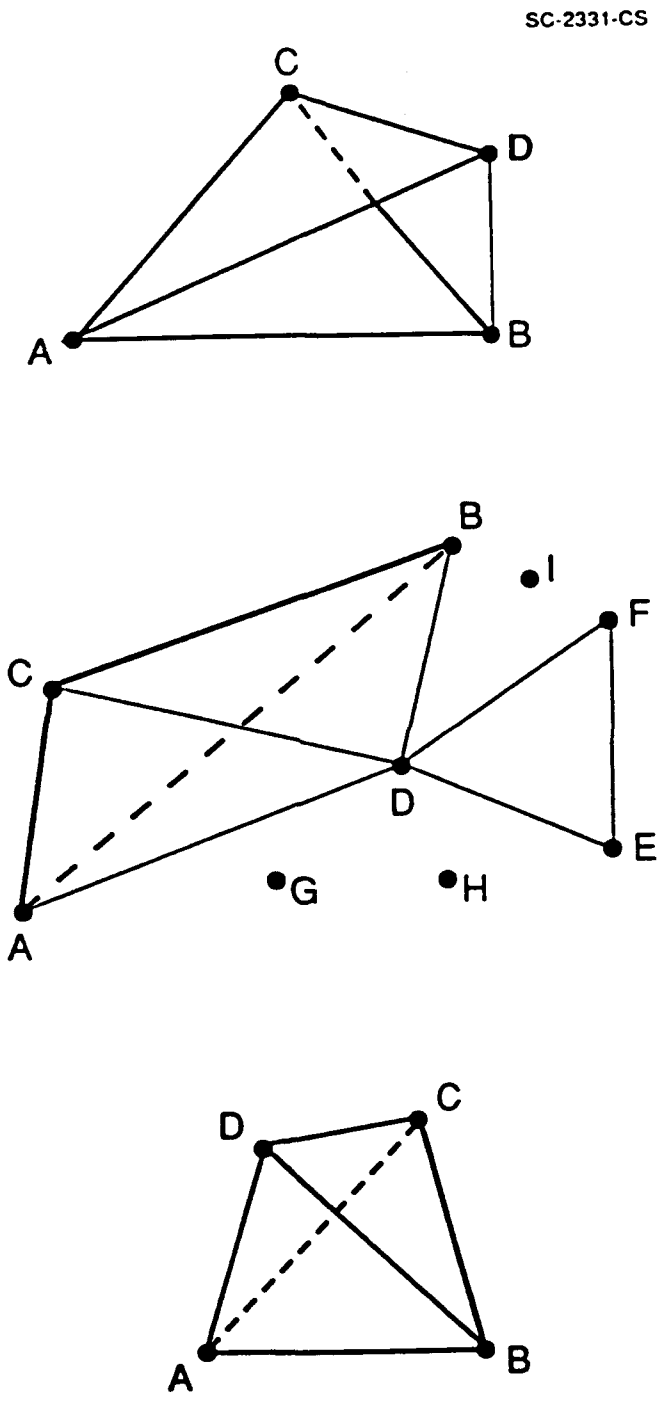


Fig. 9.6 Construction of new element

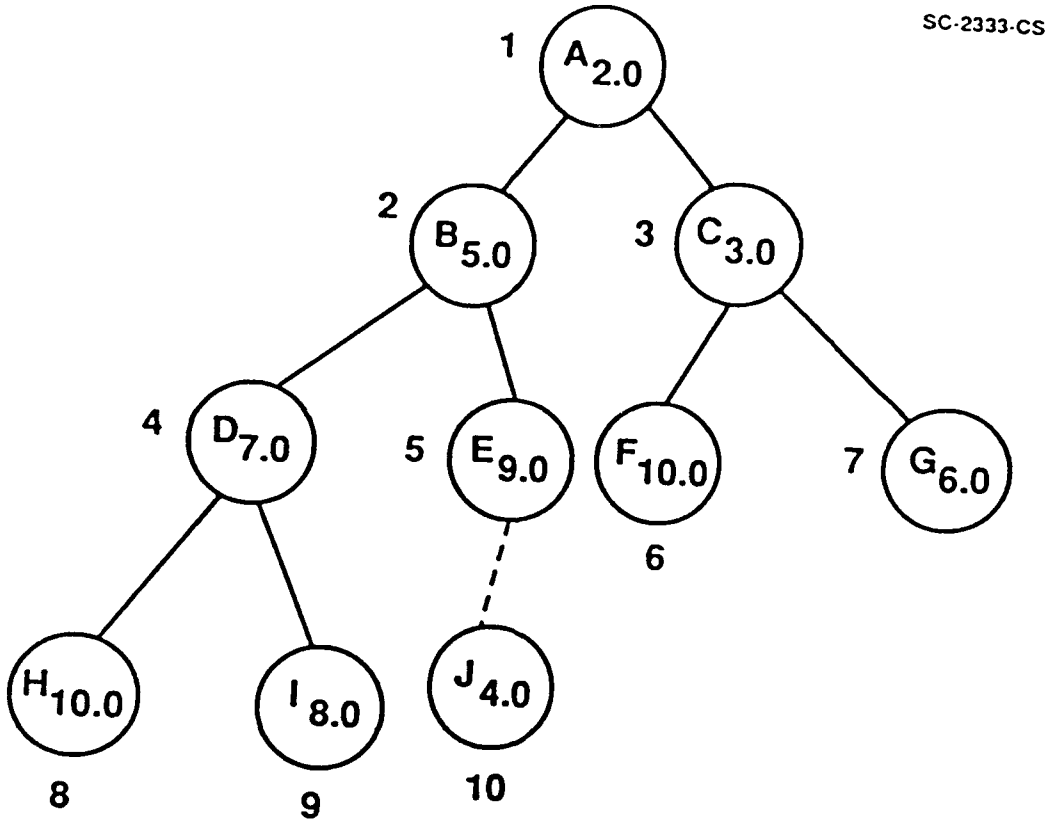


Fig. 9.7 Binary tree

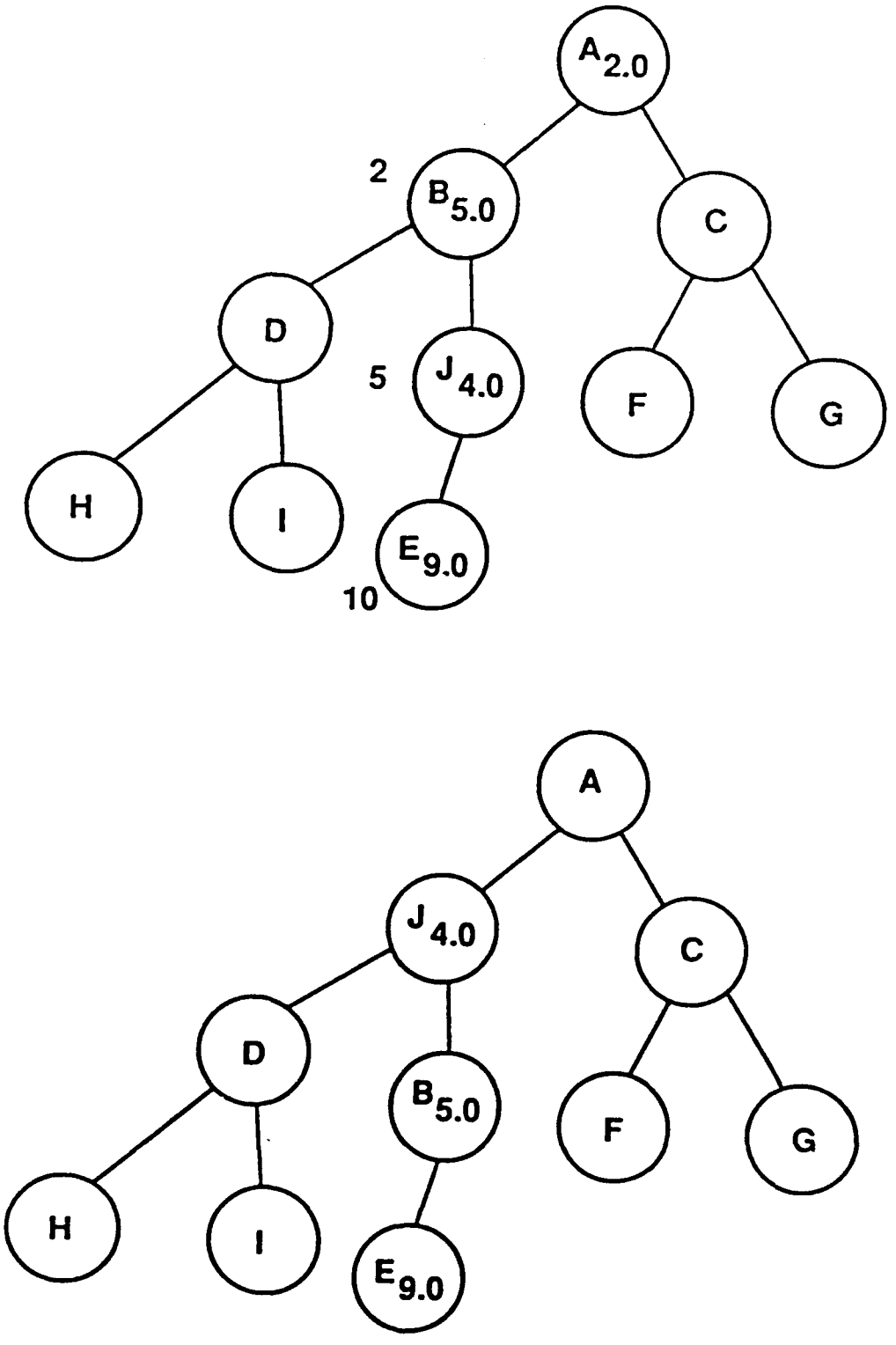


Fig. 9.8 Addition of new face to binary tree

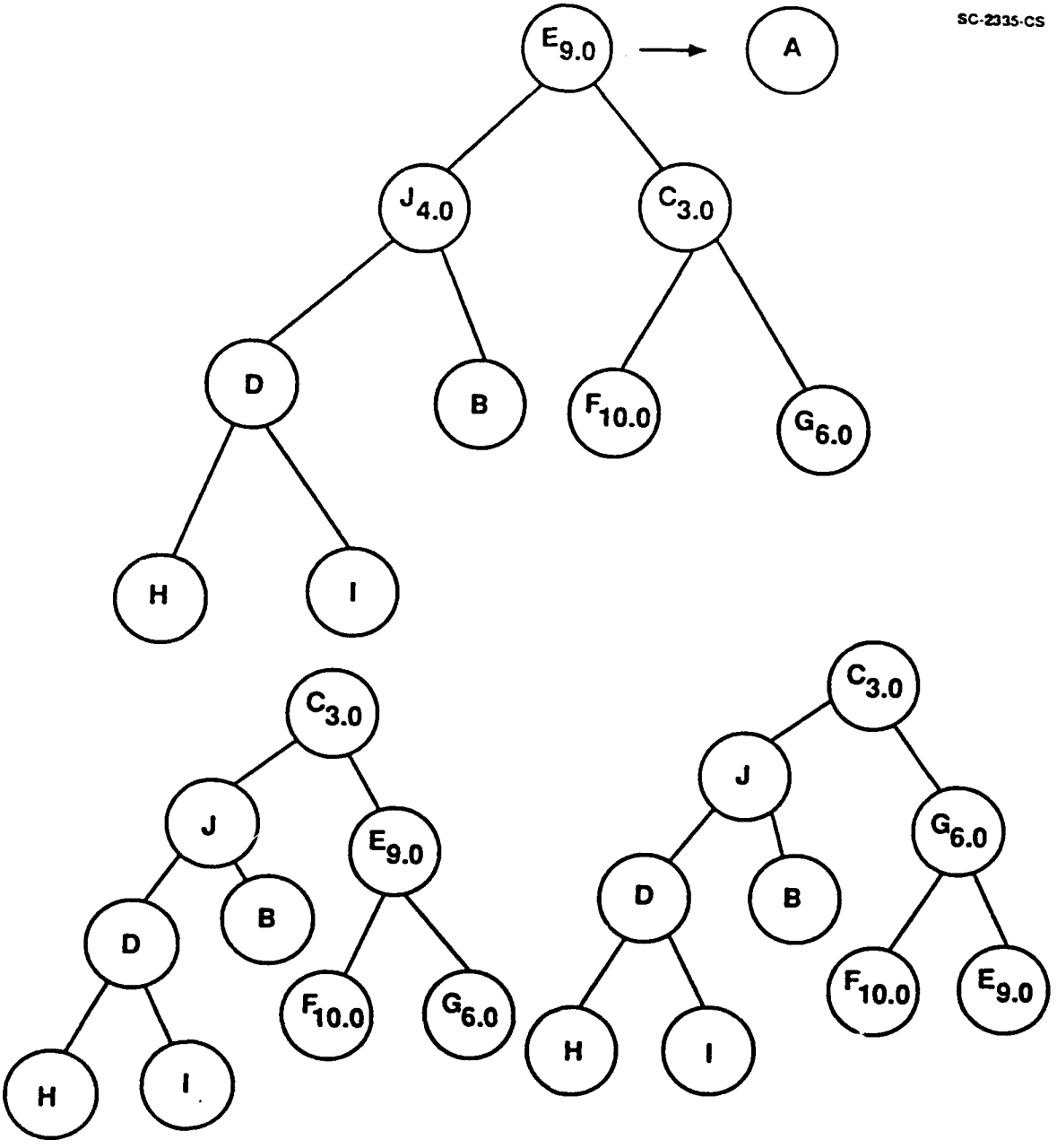


Fig. 9.9 Removing a face from binary tree

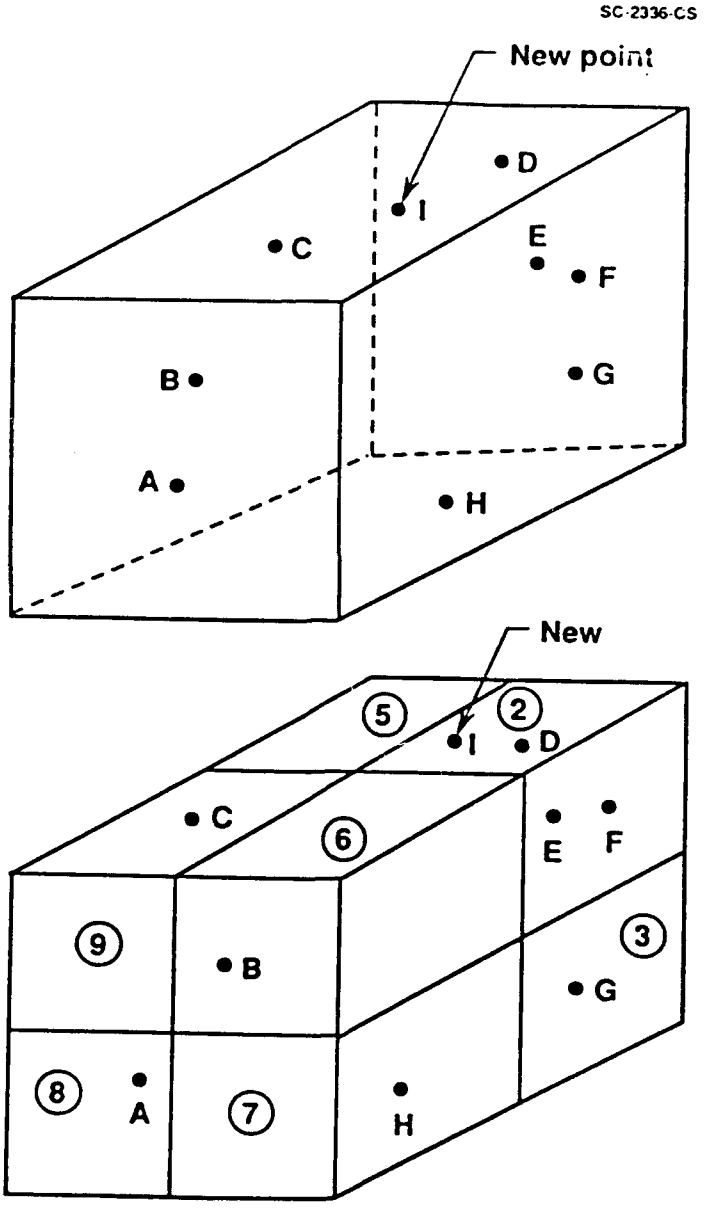


Fig. 9.10 Construction of octree

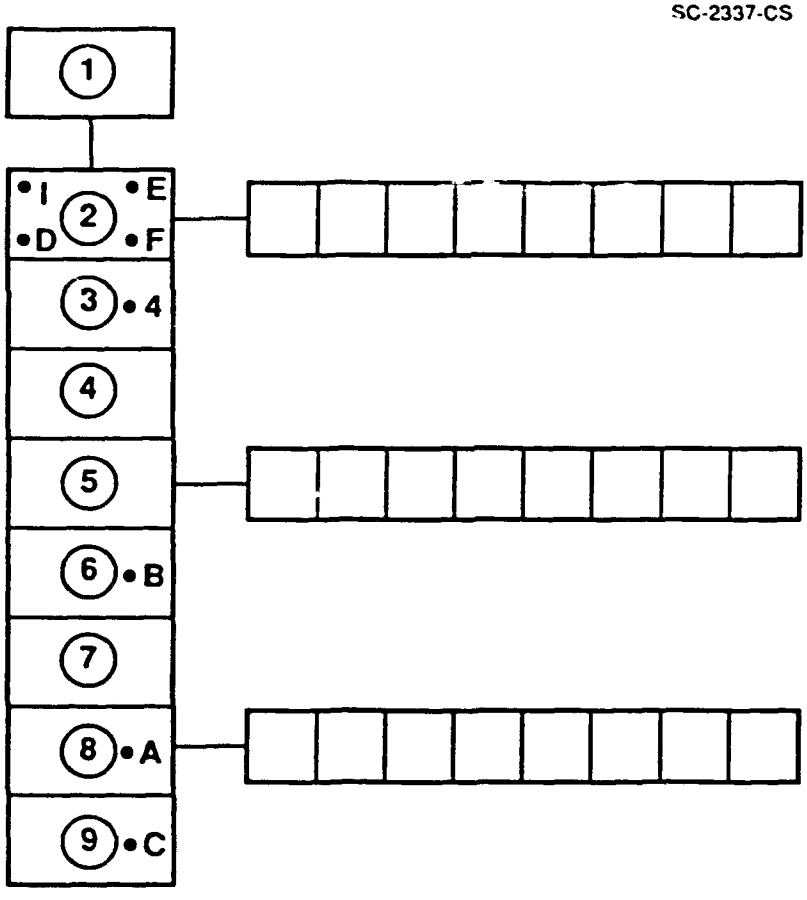


Fig. 9.10 Construction of octree — continued

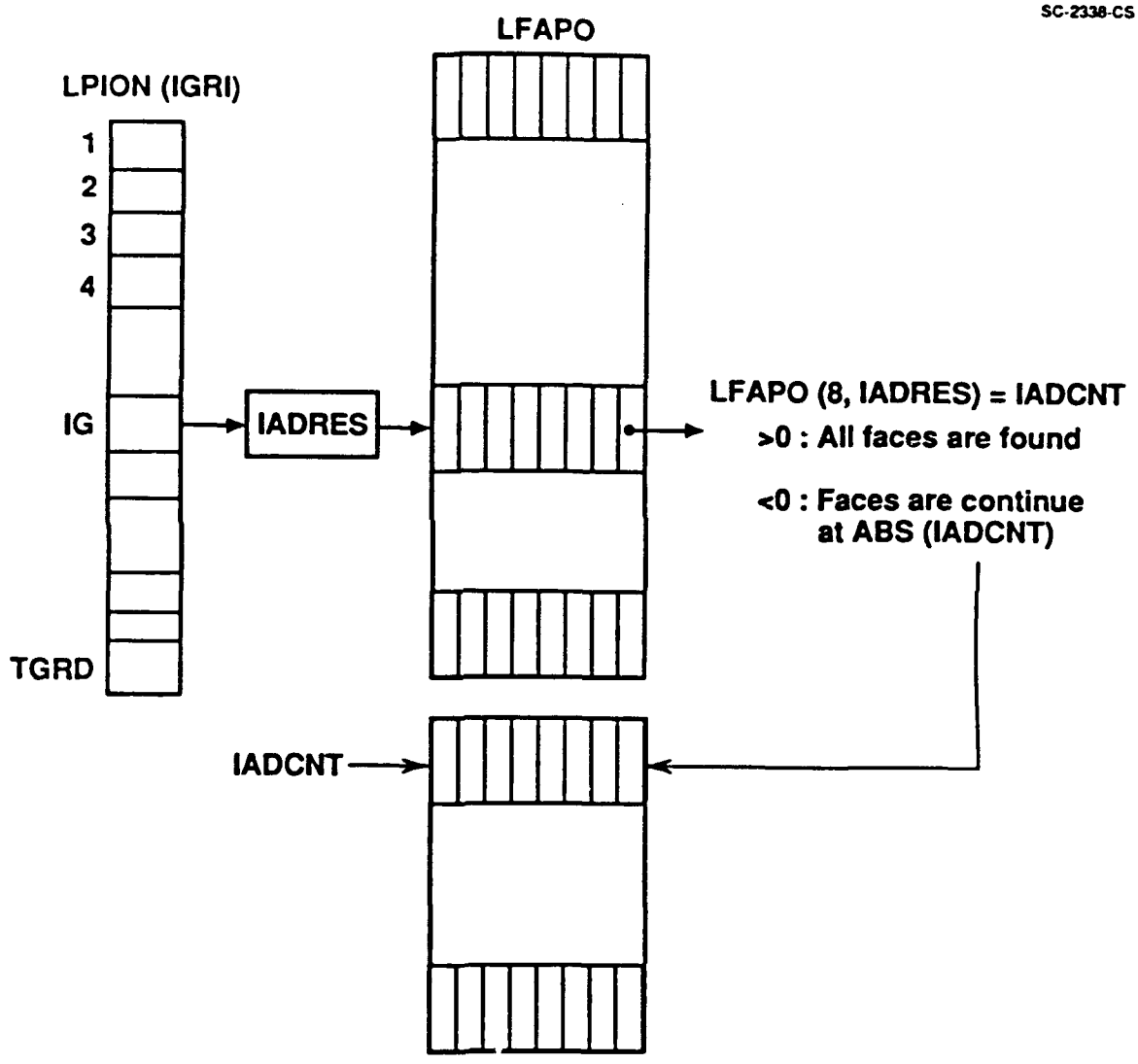


Fig. 9.11 Linked list between point and faces

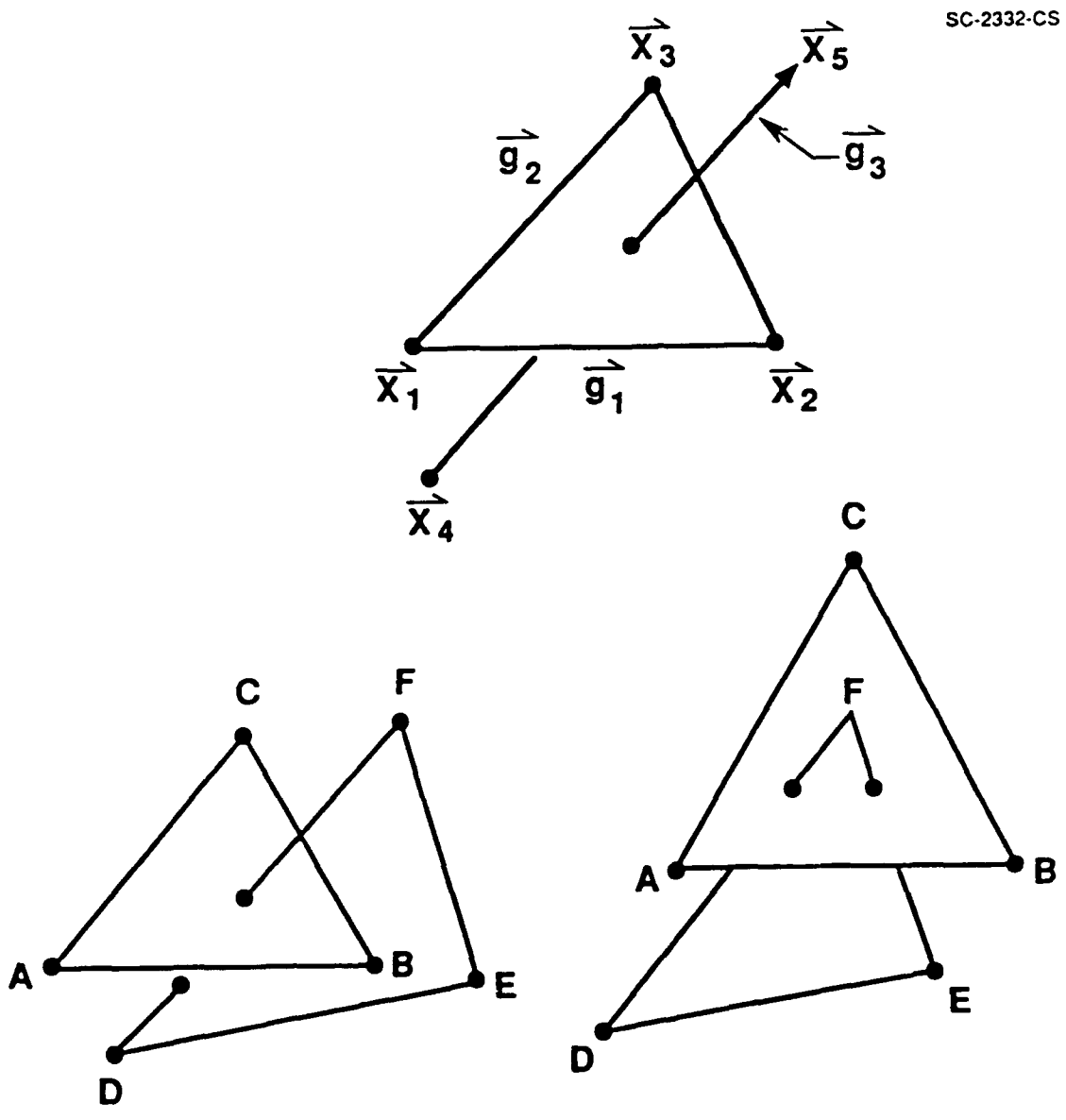


Fig. 9.12 Intersection of two faces

relative merits. Figure 9.13 shows the original mesh which was constructed intentionally to be nonuniform (with different grid densities in different regions). This was first smoothed using an algorithm by Batina.[42] In this approach based on each link being a spring, the spring stiffness has been selected to be inversely proportional to its length. The results are not very satisfactory. This was modified by assuming constant stiffness which results in the coordinate for each node being the arithmetic average of its linked neighbors. The corresponding grid is shown in Figure 9.14. The quality of the mesh is better than that of Figure 9.15. However, in general such an approach can lead to overlapped cells near leading edges and has also proven to be unsatisfactory in earlier trials in three dimensions. The third approach tried was to use a constant spring stiffness but a spring force proportional to cell volume. The resulting grid is shown in Figure 9.16. This grid is better than the previous ones. An alternate approach was developed for the fourth method. In this, instead of assuming springs along each link, a force balance was developed at each vertex based on pressure forces proportional to cell volume acting normal to each face. The resulting grid is shown in Figure 9.17. This method is also quite satisfactory. Finally, three figures are attached (Figures 9.18-20) where the diamond "airfoil" has been allowed to move over a number of time steps to various locations and orientations.

More mathematical details of the various "spring" formulations will be provided in a future edition of this report.

Higher-Order Geometry Treatment

Approaches that may be used to derive higher-order element geometries from meshes generated using TRIM2D and TRIM3D will be tried out in Phase II and will be described in a future edition of this report.

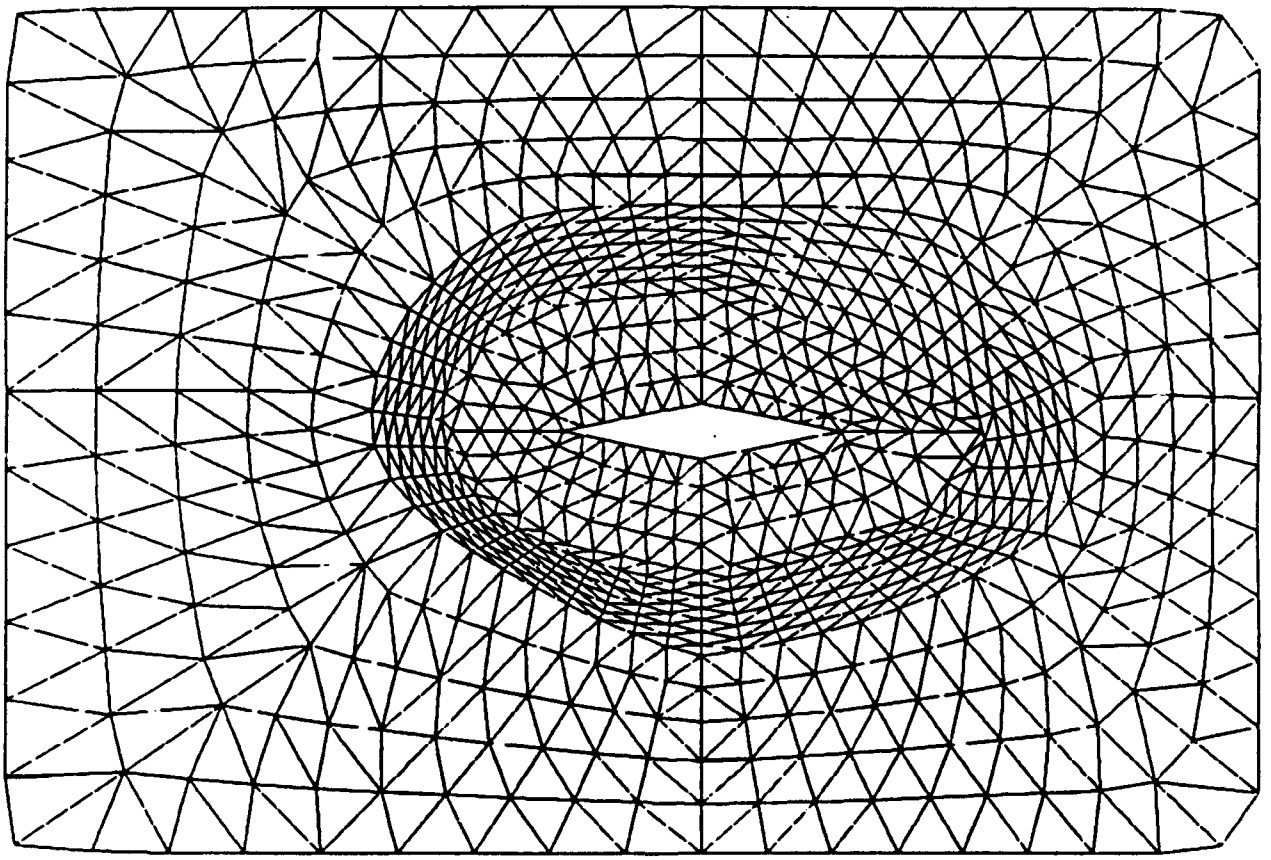


Fig. 9.13 Original nonuniform grid for use in smoothing tests

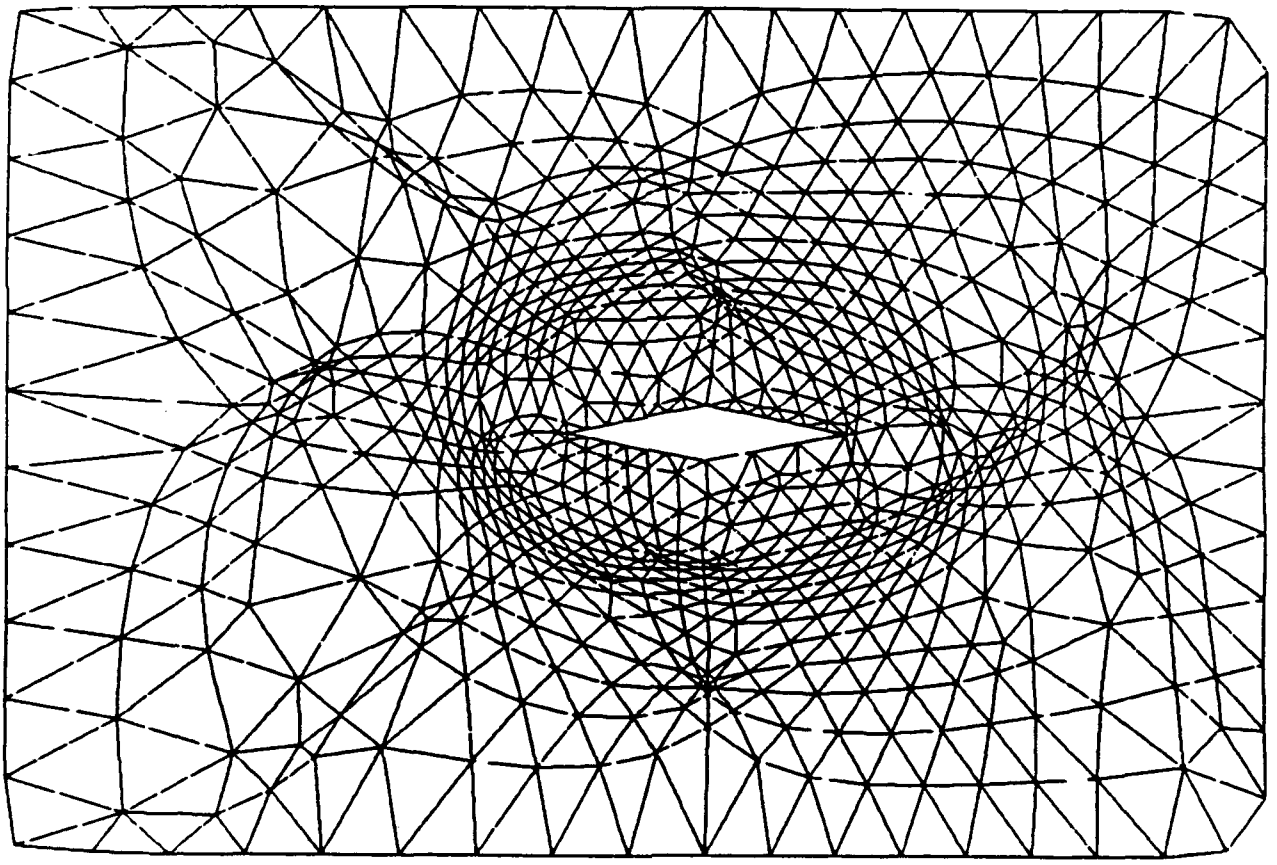


Fig. 9.14 Grid smoothed using Batina formulation

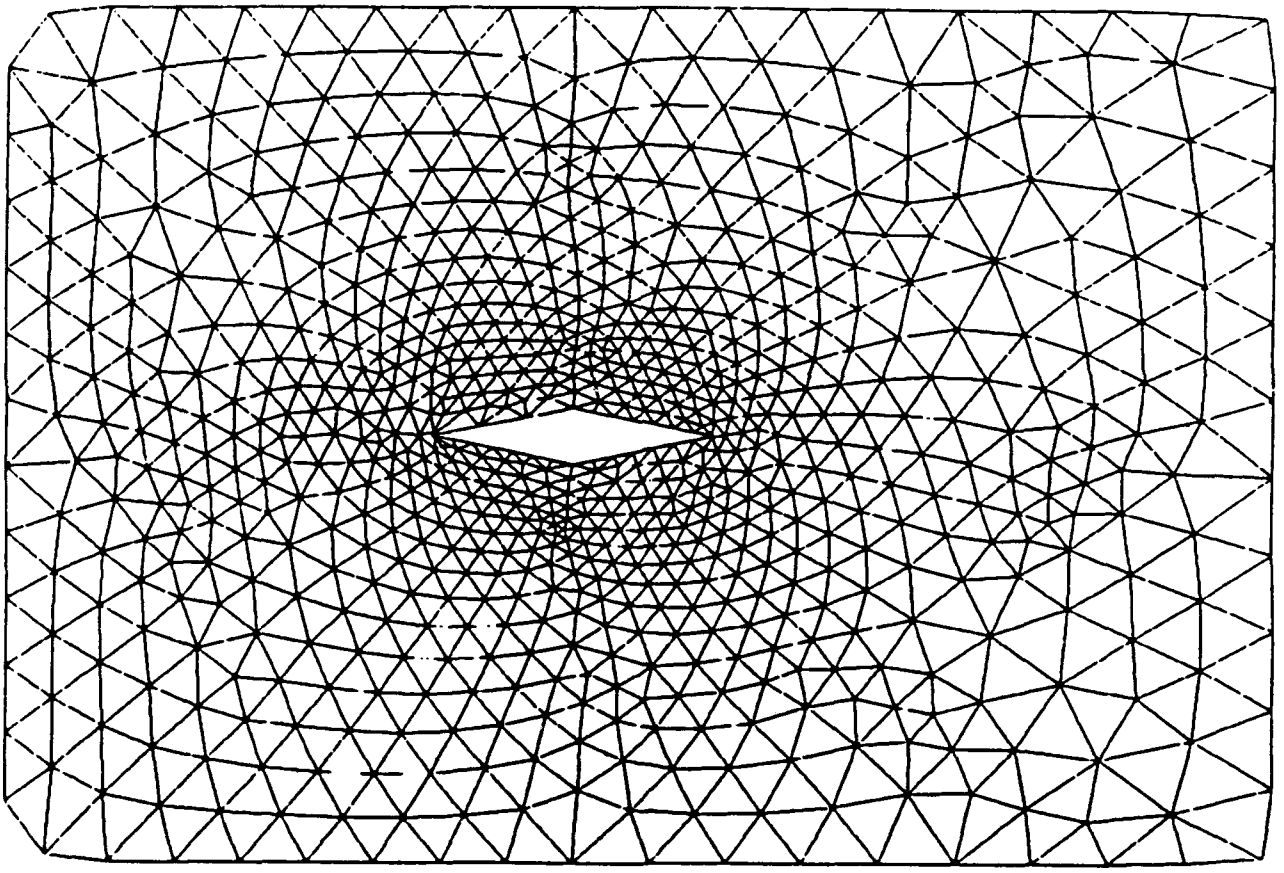


Fig. 9.15 Smoothing with constant stiffness and force proportional to length

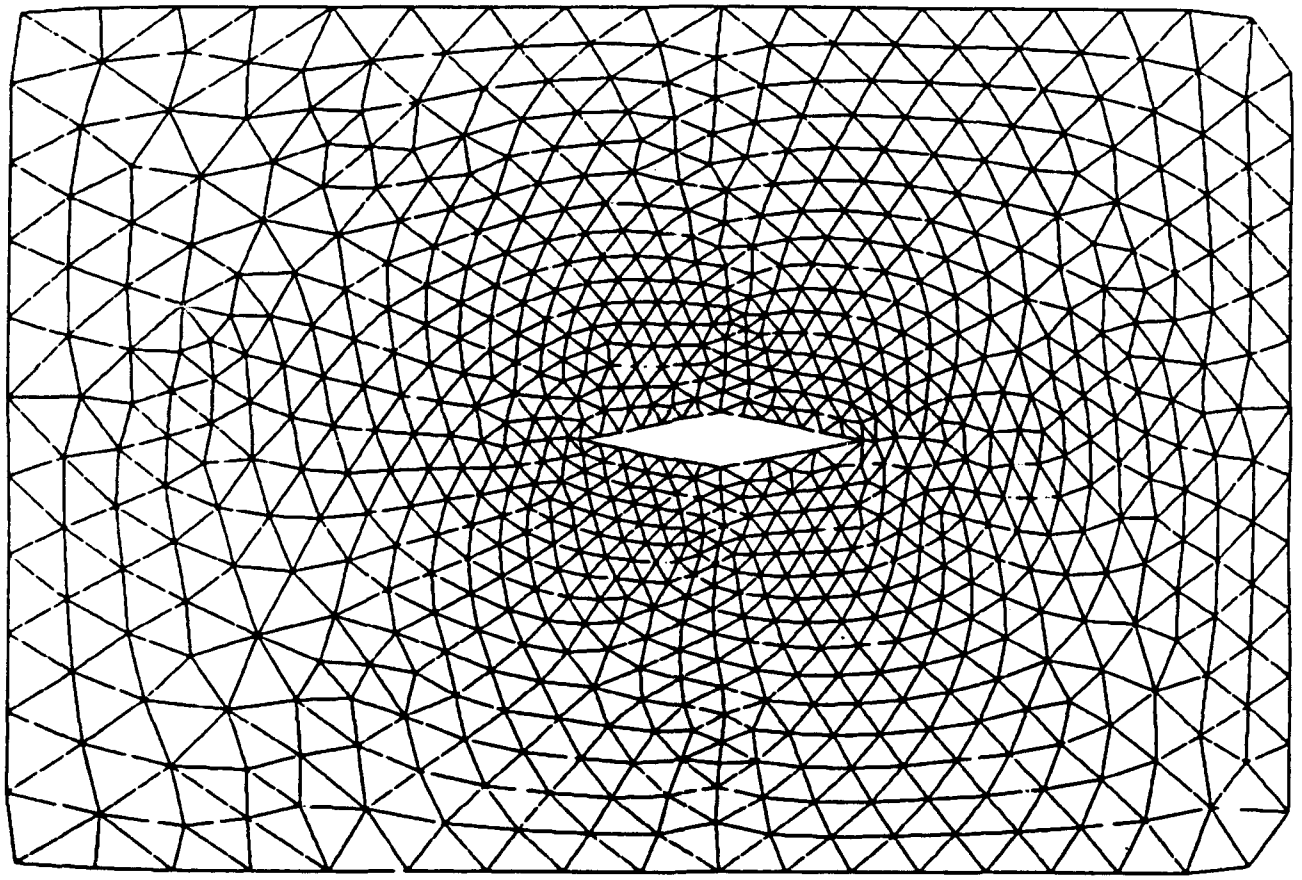


Fig. 9.16 Smoothing with constant stiffness and force proportional to volume

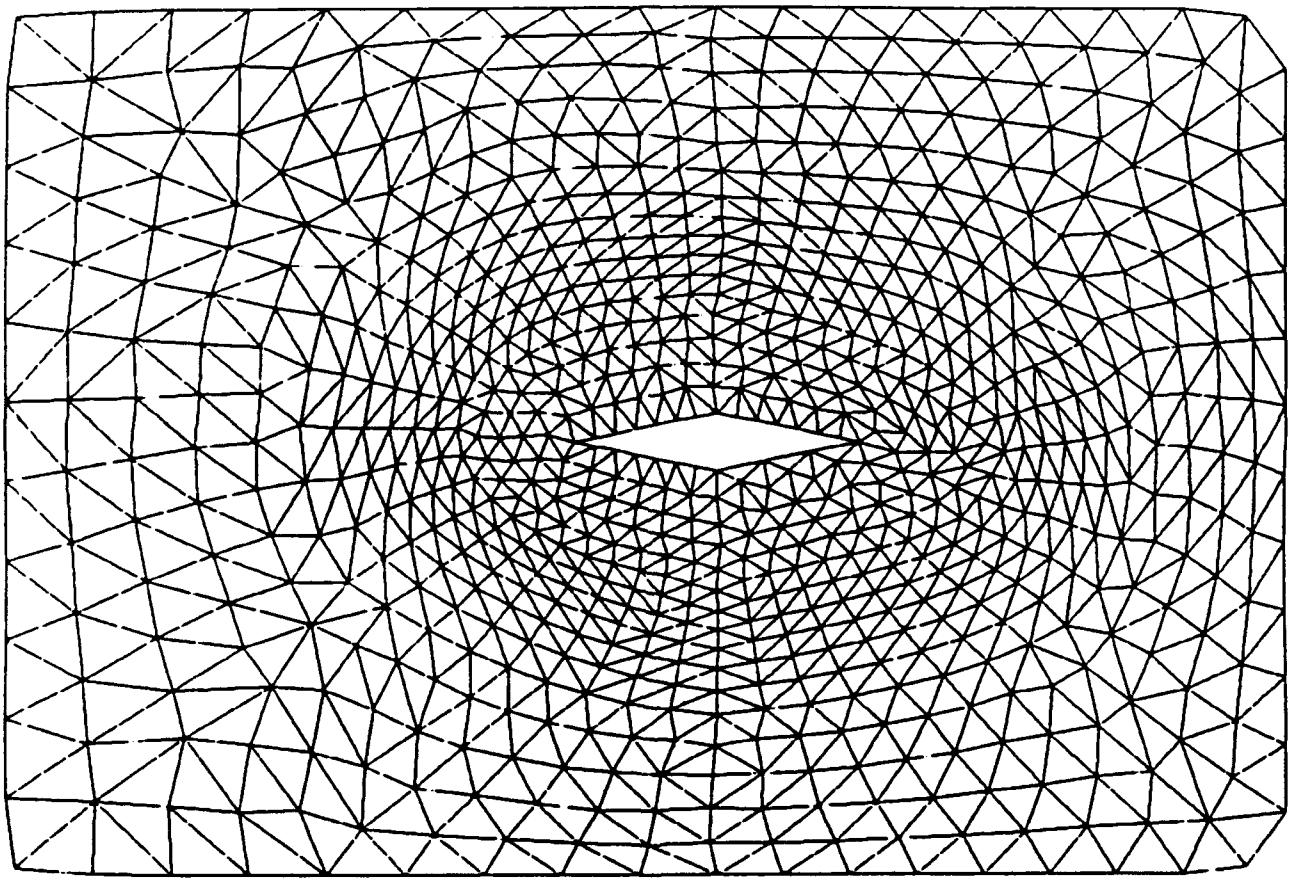


Fig. 9.17 Smoothing with "pressure" proportional to volume

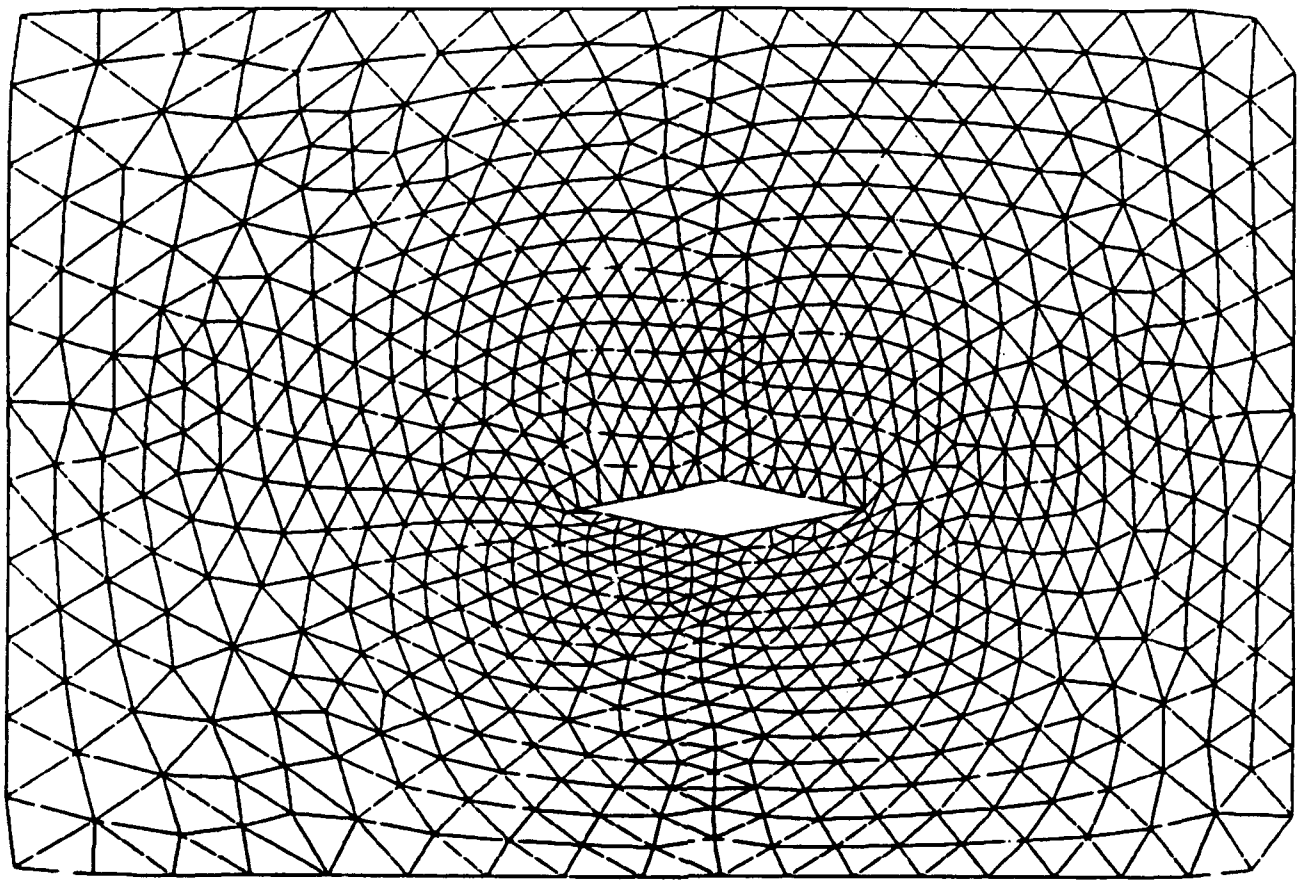


Fig. 9.18 Diamond airfoil moved down slightly from original position

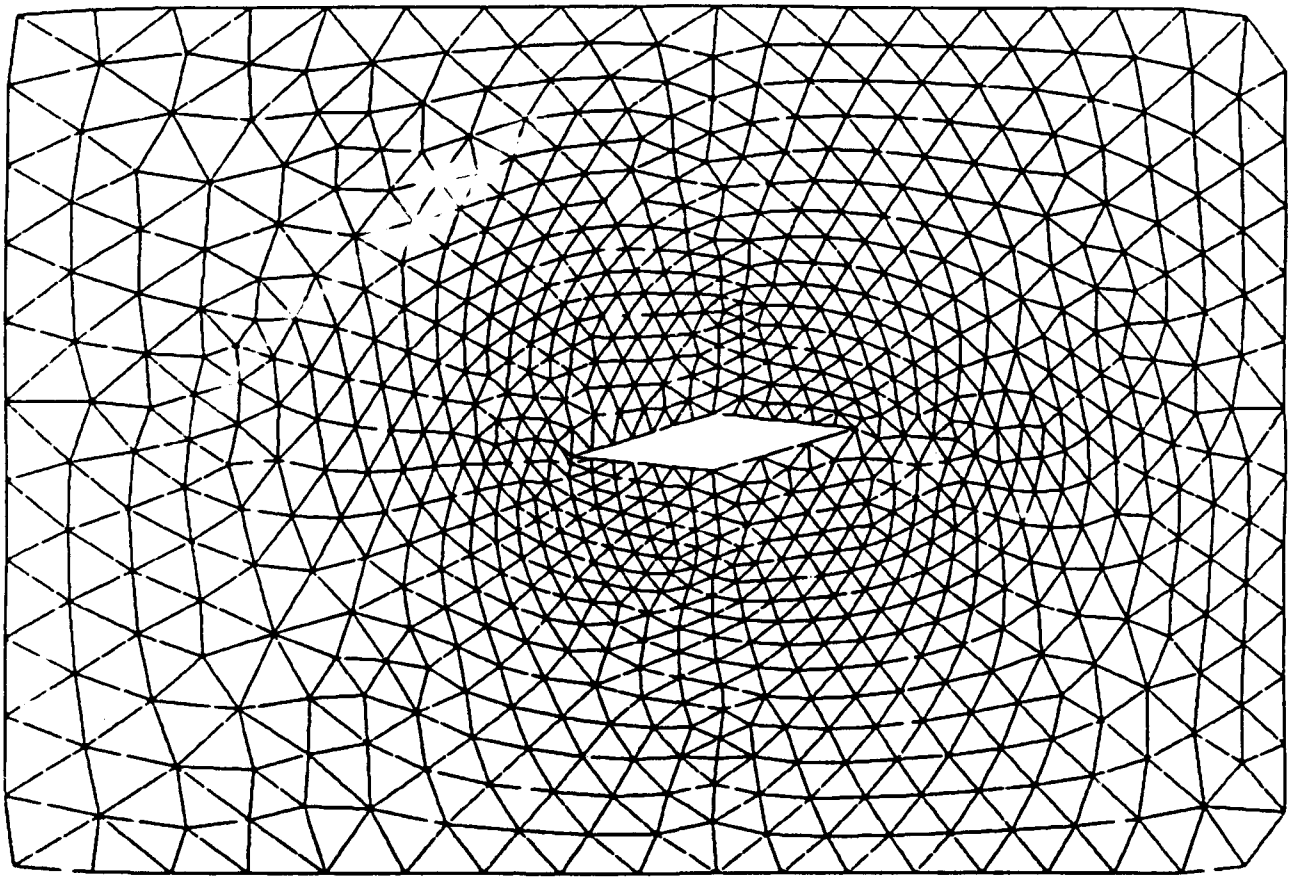


Fig. 9.19 Grid after diamond airfoil has rotated

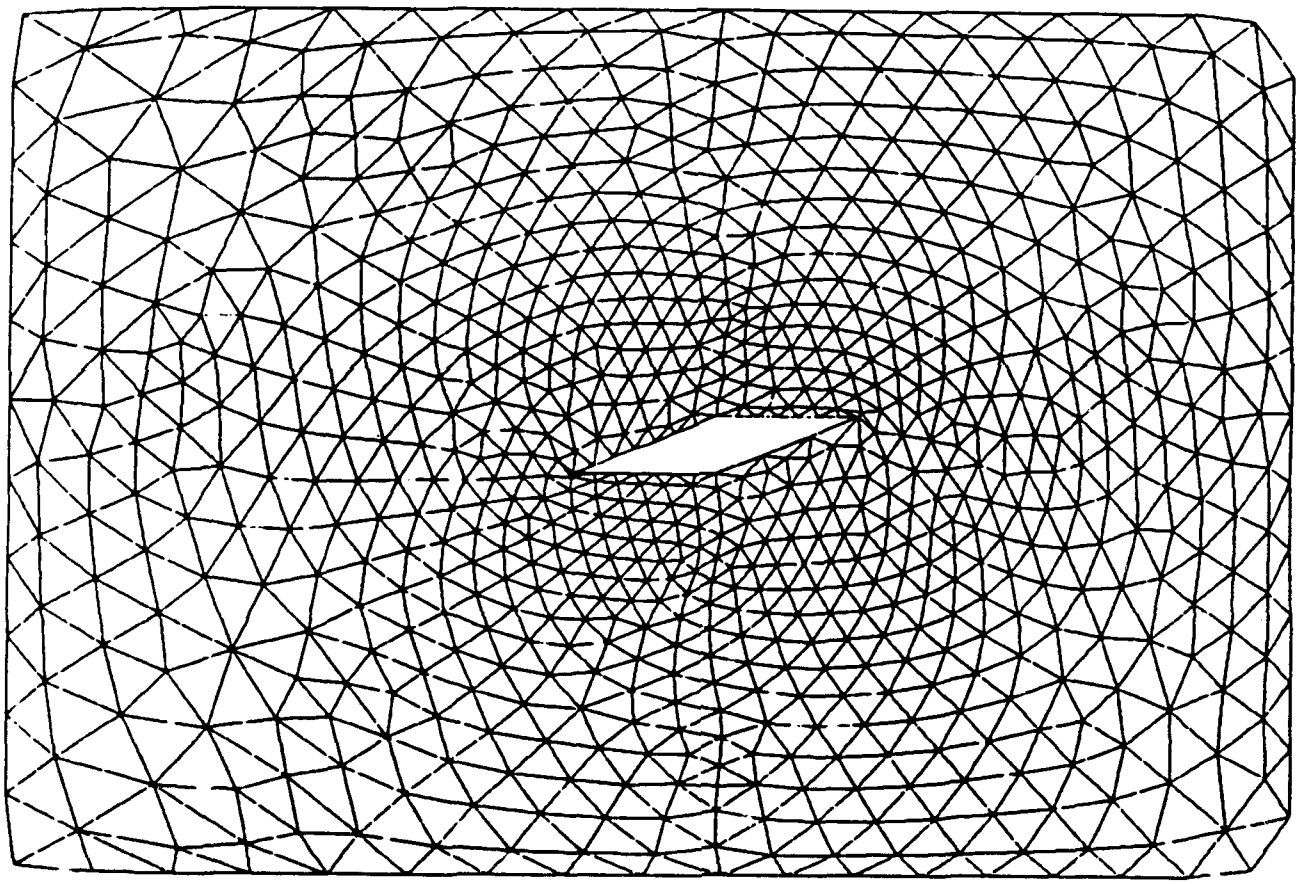


Fig. 9.20 Grid after further rotation

10.0 STORE TRACKING AND GRID ADAPTATION

We now discuss the suitability of various approaches for tracking stores separating from their parent vehicle. The weapon or other store is assumed to be initially stationary in a weapons bay (cavity) and subsequently released with a specified initial dynamics. It then moves under the influence of aerodynamic forces, gravity and its own rigid body dynamics (6 degrees of freedom). In the present effort, no propulsion effects is included. The objective is to track the motion of the store by computing the associated unsteady aerodynamic flow field coupled to a six-degree-of-freedom model for the store dynamics (Fig. 10.1).

A computational methodology developed to meet the above objective could benefit from the following attributes and capabilities.

(1) Grid point movement

(2) Regridding and interpolation of solution from one grid to another

(3) Non-overlapped patched-grid capability with sliding grid option

(4) Overset grid capability

Grid Point Movement

When the store moves due to aerodynamic loads (or motion forced on it by the release mechanism), mesh points attached to the store must move with the corresponding velocity. All mesh points in an appropriate neighborhood of the store are also moved smoothly with respect to the motion of the points on the store. This can be achieved using a method which assumes that the connection topology of the grid is a connection of springs of suitable stiffness. Mesh points far away from the moving object may continue to remain fixed in time. The node points inside this region (considered as a collection of springs) is moved in an iterative process until the spring system reaches equilibrium based on a suitable minimum-energy or variational principle. The length of a node to node connection can be used as a measure of spring stiffness and new locations of the node points can be obtained by either Jacobi

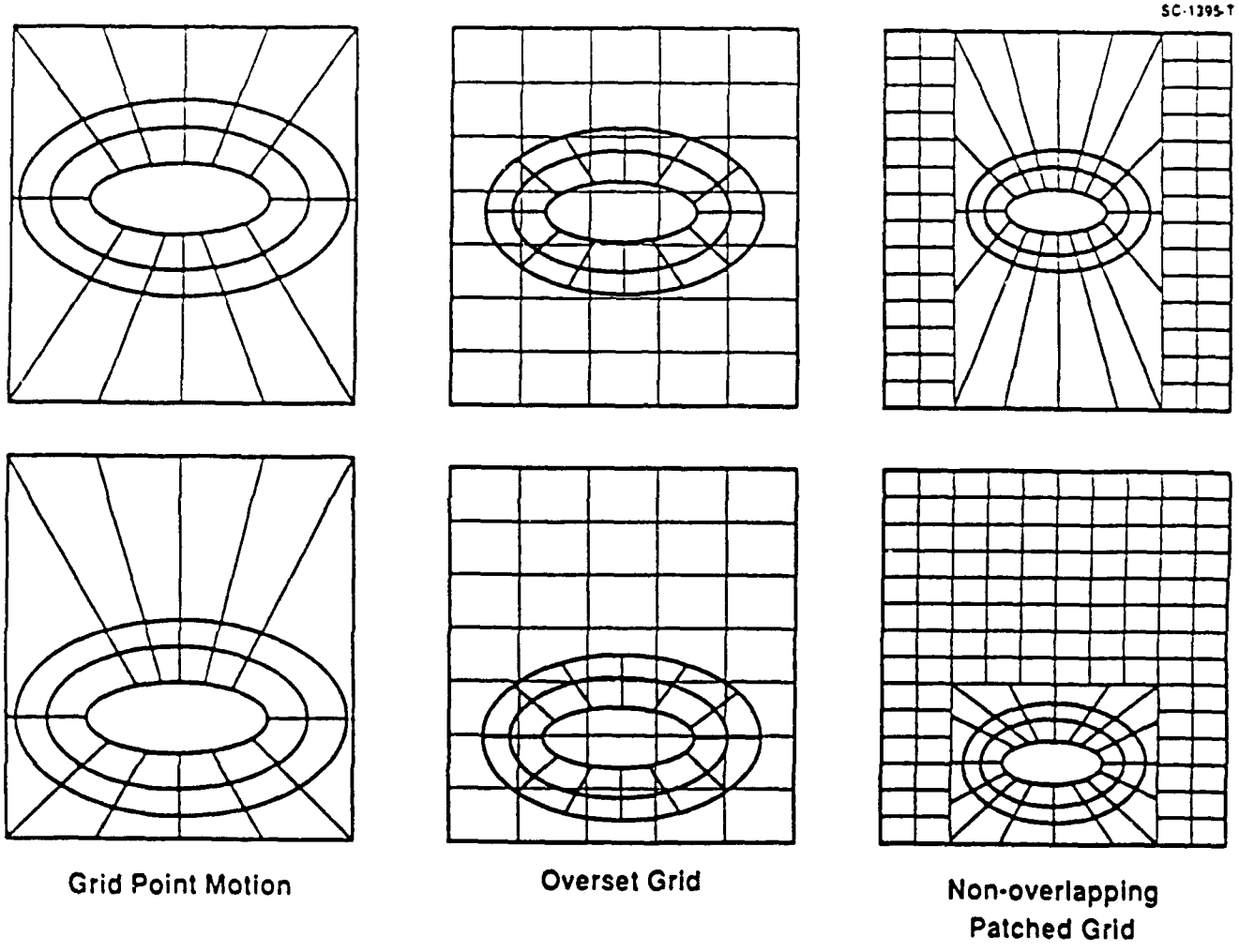


Fig. 10.1 Three types of gridding techniques for moving bodies

iteration or predictor-corrector scheme. The primary objective of this approach is to develop smooth mesh movement that is related to body surface movement. Examples of various grid smoothing ideas were provided in the previous section (Section 9) on grid generation.

Adding and Removing Cells

As the store moves, mesh points attached to the store will move with it. If we do not want to continuously smooth the mesh in the neighborhood of the store, we will encounter situations where the cells next to the store surface will become unsatisfactory. The links could become very small, for example. On the other hand, links may become very large too, depending on the local direction of motion of the store. When the angle between two adjacent faces (or lines in 2-d) becomes greater then a specified value or when a link becomes too large, a cell can be subdivided. For removing cells, the length of a link of a cell could be used as a critereon in the case of inviscid flow. For both inviscid and viscous flows, the critereon for cell division or link removal could be based on the goodness of the local numerical solution of the flow field. When the cell structure is changed, the solution variables have to be reassessed. This is done by a suitable interpolation from the cells of the original topology. The advantage of local operations such as the above is that the interpolation process is easy since no global searches will be required. In Section 8, we discussed several aspects of cell division and link removal.

Regional Remeshing

In a chosen region around the moving body all the node points may be removed and new grid is generated using the original method applied to the selected region. The solution reinterpolation process in this case is much more involved.

Grid Point Movement with Regridding Options

We first consider a combination of the attributes (1), (2), and (3). An intial grid is generated for the stationary store and the steady state solution is obtained. The initial flow-field generated by the CFD flow solver will be used to generate the

airloads acting on the store. The rigid-body dynamics of the store will be used to compute the store's response to the airloads. This will be translated into grid point velocities for those nodes situated on the surface of the store. Grid point velocities can also be assigned to all nodes in a chosen small or large region surrounding the store. These velocities are selected to be dependent in a convenient way on the nodal velocities defined on the store surface. The fluid-dynamic variables and the grid point locations are integrated in time together. This results in node point motion and the conservation cell (element) shapes will consequently change with time. After a lapse of time, the resulting cell shapes may be unacceptable. A new grid is created out of the old grid in such a way that the solution can be easily reinterpolated on to the new grid from the old. Grid point (element vertex) and cell removal and addition (grid editing) capabilities can be defined as part of the regridding strategy. Regions of the grid can slide past contiguous regions also. This sliding mesh technique has already been employed successfully using USA-series codes in the study of two-dimensional flow past a rotor-stator configuration in Ref. 34.

Overset Grids

Overset grids require information from the underlying grid at their boundaries. The information contribution from the overset grid to the underlying grid must also be determined. This could be highly involved and complicated in three dimensions. Therefore overset grid approaches will only be considered if the preferred approach mentioned above (using grid movement and regridding) is inadequate. The interpolation technique required for the regional remesh option will also be applicable to overset grids.

11.0 BOUNDARY CONDITION PROCEDURES

We now discuss some theoretical and some practical issues related to implementing boundary condition procedures in the UNIVERSE-series formulation.

Theoretical Considerations

In Section 6, we described the Riemann Initial Value Problem (IVP) and its solution. The solution involved piecewise constant states separated by transitions which are either continuous or discontinuous. The initial values for the Riemann IVP were the left and right states. Knowing the left and right states allowed us to solve the Riemann problem including all the individual transitions. We also noted in Section 6 that each transition was a one-parameter family of solutions.

When boundary conditions must be considered, we focus our attention on the Riemann Initial and Boundary Value (IBVP) problem. At a "left" boundary, for example, we assume a constant state to the right of the boundary along with boundary conditions to be specified at the boundary. Let us consider an example with a linear system of equations where there is one eigenvalue of the coefficient matrix that is positive. In other words, one wave points from the boundary to the interior. Therefore, between the given right state and the boundary, there is only one transition with one free parameter. If the boundary condition is specified so that this free parameter can be determined, the Riemann IBVP can be solved. If there are two right-moving waves at a left boundary, two suitable boundary conditions will be required.

The nonlinear case is a little more complex but can be dealt with in a fashion similar to how nonlinearities were treated in the discussion of the Riemann IVP in Section 6.

This serves as a brief but general introduction to the theoretical boundary condition framework. We postpone for the future a detailed discussion of various specific boundary conditions including surface tangency, supersonic and subsonic inflow and outflow.

Practical Considerations

A complete description of any CFD problem to be solved requires geometry definition, boundary condition specification, and initial condition specification. Typically, different boundary conditions are employed at different boundaries. One would like to have the flexibility of being able to specify different boundary conditions at each boundary point or cell face in the computation.

In structured grid multi-zonal computations, boundary conditions can be conveniently specified by "regions". A "region" of the boundary can be identified easily by its zone number, and its structured grid index limits ($JBEG(NREG)$ to $JEND(NREG)$ and $KBEG(NREG)$ to $KEND(NREG)$, for example). In structured grids, adjacent cells have adjacent values of the indices (J and K at the boundary in the example being considered). For unstructured grids, the situation is more complex. Neighboring cells are not necessarily tagged with consecutive cell numbers (this is impossible except in one-dimensional string of cells). Based on the face numbering system described earlier, this also implies that adjacent boundary faces will not have face number identifiers that are consecutive.

One easy way to tag boundary locations with boundary conditions is by using global node numbers. The geometry specification is typically done patch by patch. Node numbers in a patch can easily be numbered in consecutive order (except at a patch boundary which is common to another patch). If we want to impose one boundary condition type per patch, one can look at the boundary cell face being considered, identify the global node numbers of the vertices of the face, identify the maximal node number, and apply the particular boundary condition type associated with that node. This is the approach taken in UNIVERSE-series codes.

12.0 SOFTWARE ARCHITECTURE ISSUES

A Software Triad

A very flexible and versatile software architecture framework is being implemented in all Science Center CFD Department projects including the Store Separation contract with NWC. All software will comprise three parts:

1. User interface
2. Methodology (Physics and Numerics)
3. Visualization

Previously most CFD groups thought in terms of the three stages of preprocessing, solution runs, and postprocessing and the software was aligned with these three phases. In the emerging UNIVERSE-series codes, the aim is to think in terms of problem definition phase, solution phase and analysis phase. The solution phase will include automatic grid generation, flow simulation, grid adaptation, etc. which used to hitherto be considered separate stages. While the UNIVERSE-series software will require little or no runtime intervention on the part of the user, it is still desirable to allow the user the ability to track the solution process as it happens. In any case, in the problem definition and solution analysis phases, user involvement is a must. Therefore, in the new software architecture, all phases of CFD will include the user interface, the methodolgy and the visualization sections.

The user interface will be based on the X-window environment. A simpler and conventional approach will also be provided for use from "dumb" terminals where the user can edit a file of control variable inputs to control a subsequent CFD run. The user interface software will interact with the methodology and visualization software as appropriate.

The software segment identified as "methodology" will consititute the bulk of the computer program libraries. This will include all information about grid generation, book-keeping, solution, physics modeling, numerical modeling, etc. It will also include all software needed to generate information to be handed over to the visualization segment for graphical display purposes. For example, it will be the responsibility

of "methodology" to generate the streamlines while it will be the responsibility of "visualization" to display the streamlines via the user interface. The "methodology" segments will interact with the user interface and visualization segments as appropriate.

The visualization segment will handle the creation of graphical objects and their manipulation and display using information provided by the user interface and the methodology and will interact with the other two segments as appropriate. It is of considerable advantage to program the visualization segment using a standard graphics library such as "PHIGS+". On the other hand, the visualization segment is much smaller than what used to comprise "plotter" software. Therefore, it becomes very easy to create multiple versions of visualization software, one using the Silicon Graphics graphics library, for example, and another using PHIGS+.

In a graphics exercise for postprocessing, the user interface will obtain from the user information about what the user wants to plot, how many contour levels, viewpoint, light sources, etc. The "methodology" will provide the actual contour lines or polygons with function values at the vertices, etc. The visualization software will take all this and produce a rendered image which will go back to the user interface for display.

The three segments will be three processes running either on the same computer or even on three different computers. They will communicate via message passing using RPC protocol (Remote Procedure Call) and "sockets" which are part of the TCP/IP network protocol.

This approach will provide total consistency between methodology and graphics. Therefore much more appropriate information will be displayable including for example: piecewise polynomial behavior; right hand side values to detect location of large transients; local accuracy measures based on differences between "right" and "left" values at cell interfaces; numerical flux; geometry patch and problem specification information. This approach also minimizes the amount of work in the "visualization" segment so that it will be easy to port those modules to various hardware platforms and graphics libraries.

The approach presented will alleviate the need for large amounts of real memory,

disk space, etc. on all the computers being used. For example, let us assume that a large scale simulation is performed on a CRAY computer and the results stored on its local disks. Let us also assume that a Silicon Graphics work station is available for graphics. The user interface and visualization software will then be able to run on the Silicon Graphics work station and the information that is needed for display can be supplied over the network using the methodology program running on the CRAY. Typically, only a few planes of information from a full CFD simulation are displayed at any one time. This prevents having to bring the entire solution file from the CRAY to the work station. Of course, an even better scenario is for the user interface software to be run on an even cheaper "X" workstation leaving the Silicon Graphics with its "graphics engine" to be able to perform rotations and translations using that but passing on the pixel information back the "X" station for display.

The Use of C Language

The C language offers many attributes that can greatly help the development of very versatile CFD software as part of UNIVERSE-series code development. Since most CFD software today has been written in FORTRAN, a comparison of certain features of the two languages is presented below for review. The C language is playing a major role in the software being developed as part of all UNIVERSE-series projects including the work sponsored by NWC.

1. Greater compartmentalization:
   C is a structured language while FORTRAN is not. In C, it is easier to compartmentalize code from data. This means that all information and instructions necessary to perform a single task can be hidden from the rest of the program. One example is that local variables can be declared within any code block. Another example is that the default argument passing into functions is by value (except for vectors).

2. Local variables:
   As pointed out earlier, in C local variables can be defined within code blocks. The variables needed within a *for loop* can be defined within the loop. This assists in compartmentalization. Local variables can be *static*.

3. Variable types:

C includes variables types of *const, extern, static, register, auto. const* is like FORTRAN's *parameter. extern* is like FORTRAN's *common* but with far greater flexibility that can assist in compartmentalization (*static extern* variables are known only within the file they were defined in). *static* is like FORTRAN's *save* but since variables can be defined within each code block, *static* can result in variables having far more precise scope (locality) than is possible in FORTRAN. There are no *register* variables in FORTRAN. Other variable types that have no counterparts in FORTRAN are *volatile* and *void.* C does not have variables of type *complex.* However, CRAY Standard C includes complex variables and arithmetic.

4. Arithmetic operators:

C has a larger collection of arithmetic operators including % (modulo division), −− (decrement) and ++ (increment) which are not found in FORTRAN.

5. Bitwise operators:

C has a rich collection of bitwise operators not found in FORTRAN in the form of operators. These include left-shifts, right-shifts, and the Boolean operators of AND, OR, EXCLUSIVE-OR, NOT, etc.

6. ? Conditional:

C has a very special conditional operator using ?. An example is

$$x = 10;$$

$$y = x > 9\,?\,200 : 100$$

Another one is

$$\#\text{define abs}(i) \quad (i) < 0\,?\,-(i) : (i)$$

7. Pointer variables and arithmetic:

C includes pointer arithmetic. Standard FORTRAN77 does not. C has included

pointers from the beginning. C pointers include the ability to define pointers to pointers and even pointers to functions. In FORTRAN also, there is a feature similar to pointers to functions so that we can pass the name of a function as an argument to a subprogram.

8. Arrays:
   Both C and FORTRAN support unidimensional and multidimensional arrays. Some differences are noteworthy. FORTRAN arrays are stored in memory by columns. C arrays are stored by rows. C array references assume that position 0 is the first value along each dimension. In FORTRAN, the default first position is identified by a subscript value of 1. In FORTRAN, the beginning and ending values of each subscript can also be specified as part of the dimension statement. This can be achieved in C but only indirectly. The syntax of array specification is also different for C and FORTRAN. FORTRAN uses "DIMENSION $A(10,20)$". C uses "double $a[20][10]$". In C pointer arithmetic can be used to conveniently handle array operations and often pointers and arrays are intermingled.

9. Compile time *sizeof* function:
   C includes a compile-time *sizeof* function which can greatly enhance portability of the code.

10. The comma:
    C includes very flexible statement formats. One special punctuation mark is the ",". The "," operator can be used to string together several expressions. When used on the right hand side of an assignment statement, the value assigned is the value of the last expression of the comma-separated list.

$$x = (y = 3, y + 1).$$

    In C, many variables can also be equated to a single expression.

11. Structures:
    C includes the ability to define "structures" which are a very powerful data formats.

12. Casts:
    C includes a quite formal way to perform type conversions.

13. Shorthand notation:
    C includes some shorthand operators like "+=". This is not a particularly great advantage. The increment and decrement operators discussed earlier are more important features because they can lead to more efficient instructions being generated by the compiler.

14. Conditional blocks:
    C includes the standard "if" and "else" constructions.

15. Switch:
    C includes the "switch" feature. The "switch" is rather like the computed goto statement in FORTRAN.

16. Loops:
    C has three loops, 1) the *for* loop, 2) the *while* loop and 3) the *do while* loop. FORTRAN only has one type. The first two may not execute even once (like ANSI-standard FORTRAN77 loops) while the last will always execute the loop body once.

17. Return:
    C includes the "return" statement similar to FORTRAN except that in C this statement can be used to return a particular value.

18. Jumps:
    C does not exclude jump statements like "goto" and "break". The "continue" statement in C loops skips to the next iteration of the loop and is therefore not a dummy statement like in FORTRAN. In C, one can even jump from the middle of one function to the middle of another (using "setjmp" and "longjmp").

19. Exit:
Through the "exit()" function, the C program can pass an error code and control back to the operating system.

20. Dynamic memory allocation:
Standard C includes dynamic memory allocation (from the heap) routines like "malloc", "calloc", "realloc" and "free".

21. Command line arguments:
Command line arguments are standard part of the C language. A C program can use "argc" and "argv" very effectively.

22. Recursion:
On every computer system, C is a recursive language. A function can call itself. This can help add readability to a program. When parallel computers came on the scene, FORTRAN compilers had to be made "stack"-based and recursive. C was recursive from the beginning.

23. Union and Enumeration:
Along with "structures" discussed earlier, C includes other complex data types called "unions" and "enumerations". *union* is like *equivalence* in FORTRAN.

24. Bit fields:
Bit fields can be defined for "structures". This can greatly reduce memory allocations by giving us the ability to pack bit-field information into a single word rather than using a word a byte.

25. Typedef:
The "typedef" construction allows us to define new names for existing types. This enhances readability of source code.

26. Flexible I/O:

C includes very flexible I/O features. This includes the ability to perform random I/O on binary files. Both buffered and unbuffered I/O can be performed. Stream flushing is standard part of the language. Even standard input and output can easily be redirected. Input/output operations can also be done with main memory used as the buffer area (e.g. using functions "sscanf" and "sprintf").

27. Preprocessor directives:

A very useful set of preprocessor directives is standard part of the language. Macro definitions, conditional compiling, etc. can easily be accomplished. ANSI C includes the "#pragma" directive which is used to define machine-dependent treatments like vectorization, parallization, etc. This is similar to the "CDIR$" compiler directives on the CRAY computer for FORTRAN programs.

28. Built-in debug primitives:

C includes "__LINE__", "__FILE__", "__DATE__", "__TIME__", and "__STDC__" variables which may be exploited by the programmer to provide debug information in a very portable fashion. For example, the "__LINE__" predefined macro contains the line number of the currently compiled line code. The "__FILE__" identifier is a string that contains the name of the source file being compiled. These two can be reset using the "#line" preprocessor directive. The "__DATE__" macro contains the date of translation of source code into object code and the corresponding time is in "__TIME__".

29. Comments:

C offers a much more flexible commenting capability than FORTRAN. For example, a C comment can be an embedded part of any line.

30. Errors:

Execution errors can easily be identified using "errno" which is a global variable and the function "perror".

31. Signals:

C's "signal()" function may be used to specify the function to be executed if the specified signal is received. The signal could be for abort termination, errors in arithmetic, invalid function, system interrupt, invalid memory access, external termination request, etc.

32. Unique temporary files and filenames:

C includes standard functions to access a temporary file and create a temporary file name.

33. String and character manipulation:

The C library includes a very powerful set of string and character manipulation functions. If our programs should include an ability to parse input expressions, FORTRAN would prove to be very clumsy for that purpose and one would have to employ C.

34. Exit processing:

C includes a very powerful exit processing capability using functions such as "atexit()" and "abort()". This can also assist in developing very user-friendly software because the computer program can be written to provide internal trace-back and diagnostics.

35. Sorting and searching:

C includes library functions to perform "quick sort" and "binary search" operations on records.

36. System interface:

C includes a library function "system()" which can be used to run any system command. Along with "getenv()", etc. C can easily be interfaced with the operating system. Remember that the whole UNIX operating system was written

in C. C provides many standard system interface functions related to "time" and "date" information.

37. Mathematical functions:

C includes the usual mathematical functions including the trigonometric functions, hyperbolic functions, exponential and logarithmic functions and some miscellaneous functions. C includes a random number generating routine. C does not have standard library routines to handle *complex* variables because C does not include variables of type *complex*.

38. Consistency checks:

The C compiler performs many consistency checks assisted by the fact that all variables have to be declared and typed. ANSI C also includes a feature called function prototyping which assists in checking consistency between function calls and function declarations. Most UNIX operating systems include a utility called "lint" which can be used to detect features of the named C program that are likely to be bugs, to be non-portable, or to be wasteful. It also performs stricter type checking than does the C compiler. The FORTRAN equivalent of "lint" is available but not standard and therefore is often not found in the standard set of utilities on all installations, with UNIX or otherwise.

39. C and FORTRAN coexistence:

In most modern environments, especially those that use the UNIX operating system, C and FORTRAN routines can be linked together to produce a single executable. This is particularly easy when self-contained subprograms are being linked. FORTRAN "COMMON" blocks are somewhat more tricky to refer to from C but I am aware that atleast on CRAY computers running UNICOS, C externals can be referred to from FORTRAN programs. Character variables are also treated differently in C and FORTRAN. There are also natural incompatibilities when it comes to data types not available in one language or the other.

40. Main program:

FORTRAN main program can be named anything by the programmer. The C main program must be called "main". It has already been pointed out that command line arguments are passed to the C "main" program automatically and can be used when desired.

41. Variable length and type parameter lists:

In C, you can specify a function that has a variable number and type of parameters. For example, this declaration specifies that **func()** will have at least two integer parameters and an unknown number (including zero) of parameters after that.

```
func (int i, int j, ...);
```

The three dots denote the presence of variable length arguments. FORTRAN does not have such a feature.

42. Length of function and variable names:

In C, an identifier is a sequence of letters and digits. The first character must be a letter; the underscore counts as a letter. Upper and lower case letters are different. Identifiers may have any length, and for internal identifiers, at least the first 31 characters are significant. Internal identifiers include preprocessor macro names and all other names that do not have external linkage. These features can be used to write software that is very clearly understandable.

44. Variable length arrays:

At this moment, ANSI C does not permit variable dimensioning of arrays. This is merely an annoyance because of C's pointer capabilities. However, CRAY Standard C permits variable length arrays. This feature has also been proposed as an addition to the Standard through the Numerical C Extensions Group (NCEG).

44. Summary:

C is a very flexible and powerful language. ANSI standard C has far more

standard features than ANSI standard FORTRAN. On every machine, C is a recursive language. On every machine with ANSI C, it is easier to run the same program without making any changes, even when the program includes many sophisticated constructions. For example, we have only one C version of a utility called "utof" that converts a large "update" source file into individual ".f" files and simultaneously produces the corresponding "makefile" with all the proper dependencies. On each machine that we use, we used to have a different FORTRAN version. Another example is the "update" emulator written by Rockwell NAAO personnel. It is far more easy to write portable code in C than in FORTRAN. C has all the usual features available in FORTRAN except complex variables and arithmetic. C has built-in memory allocation functions, built-in system interfaces (especially to UNIX), a very flexible debug capability that can be exploited by us, pointer capability, etc. When you look at C, you are looking at a language powerful enough to write the whole operating system (UNIX). Can we say the same for FORTRAN?

Common I/O for C and Fortran

It is equally easy to read or write ASCII (text) files using C or FORTRAN. The one difference is that in "free" format, the C language recognizes only "white space" as a delimiter between successive entities and does not recognize a "," also for the purpose. FORTRAN has a built-in record-oriented sequential binary I/O capability which adds to each record information regarding record length. The C language does not. While it is very easy to do very sophisticated file manipulation in C, without the record-oriented I/O features, we decided to develop a C language capability that mimics the FORTRAN record-oriented sequential I/O. In this fashion, users are able to write out grid files using standard FORTRAN constructions and the flow solver can read the grid in as if the computer code was written in FORTRAN also. Similarly, disk output produced by the UNIVERSE-series flow solver can easily be read in for post-processing purposes by user-developed codes which are written in FORTRAN.

Integrated Debugging and Diagnosis

The availability of macro definitions as part of the standard language makes it very easy to develop an integrated debugging and diagnosis capability into software such as UNIVERSE-series codes that is written in C. For example, test cases and even test routines can be kept part of the source code in "dormant" mode by enclosing the sections with suitable logic controlled by macro definitions. For example, if a section of code is surrounded by "#ifdef IFDEBUG" and "#endif", only when IFDEBUG is defined before this section of code using "#define IFDEBUG" will this section even be compiled. This is far more convenient than commenting and uncommenting large sections of code, a trick that is often used by FORTRAN programmers. The availability of "__FILE__" and "__LINE__" also helps in providing file and line numbers to the user when an improper condition is encountered during a run and detected by code logic. This feature greatly helps in reducing software development time also.

Dynamic Memory Allocation

We have already mentioned the dynamic memory allocation capability available in standard C. If there is one feature that must be singled out to demonstrate how the C language offers great benefits as the language of choice for writing UNIVERSE-series codes, it is this. UNIVERSE-series unstructured grid codes exploit this feature to (1) allocate only as much memory as is needed, (2) allocate it automatically based on information regarding mesh size and run options supplied by the user, (3) increase or decrease the storage automatically depending on whether new cells and nodes are created or existing cells and nodes eliminated, (4) automatically allocate temporarily needed memory space, etc.

13.0 DISCUSSION OF OPTIONS AND FEATURES

This section provides a compendium of options and features available in the UNIVERSE-series unstructured grid flow solver from the user's point of view. This section will be revised from time to time to keep pace with the current status of the code and this edition is therefore current as of November, 1991.

Various Equations

The present version of the unstructured-grid UNIVERSE-series code (UNIVC) includes the ability to solve either the linear wave equation, the inviscid Burgers' equation, or the Euler equations for compressible flow.

Riemann Solvers

For the linear wave equation and the inviscid Burgers' equation, only one choice is provided corresponding to the exact Riemann Solver. For the Euler equations, five choices are provided: (1) Rusanov scheme, (2) Godunov scheme (exact), (3) Roe's scheme, (4) Osher's scheme, and (5) Harten-Lax scheme.

Types of Cells

Three types of cells can be selected: (1) hexahedron, (2) triangular prism, (3) tetrahedron. The formulation can automatically handle degenerate versions of the first two types of cells. For example, if one face of a hexahedron collapses to become a point, a pyramid with a quadrilateral base results.

Flux Masks

An option to not compute fluxes for certain faces of each cell can be exercised if those fluxes are redundant for the particular computational simulation. For hexahedral cells, each pair of faces can be selected (corresponding to the local ξ, η or σ coordinate). For triangular prisms, the σ direction faces (triangular faces) can be

masked off or the three quadrilateral faces can be masked off. For tetrahedral faces, all faces can either be masked off together or not at all.

Spatial Accuracy

A spatial accuracy of upto 6-th order is available for one-dimensional problems. Multidimensional problems can take advantage of accuracies of upto 4-th order. The degree of the actual interpolation polynomials used to achieve these accuracies is one less than the order of the accuracy specified. The dimensionality of the problem can be specified as 1-d, 2-d or 3-d. For 1-d problems, the solution polynomial is expressed in terms of x only. For 2-d polynomials, x and y are used and for 3-d polynomials all spatial variables (x, y and z) are empolyed.

Temporal Accuracy

Time accuracies of order 1, 2, or 4 are selectable. The first-order scheme is based on the Euler explicit formulation. The second-order scheme is based on the second-order Runge-Kutta formulation and the fourth-order method is based on the standard fourth-order Runge-Kutta formulation. In the future, methods based on Taylor-series expansion in time may be employed.[19]

Quadrature Options

Integration of quantities over a cell face is replaced by a suitable numerical quadrature. This is needed for both geometric quantities as well as the dependent variables. A four-point quadrature is always used for integrating geometric quantities unless, a simpler formula is applicable without any loss of accuracy (see Section 7 for details. The user is allowed to specify the quadrature formula to be used for integrating the fluxes of solution variables. The midpoint rule can be selected. This would imply the use of one value of flux per face and one value of cell-face normals. An average value of the cell-face normal (evaluated using the appropriate geometry-variable integration procedure as outlined above) is used for the purpose. Another option is to use multi-point quadrature together with this average value of metrics. This does not cause any loss of accuracy as long as the cell face is planar and not

curved. The last and most sophisticated option allows the local value of the cell-face normals to be evaluated at each of the multiple quadrature points.

Selection of Neighbors

An underlying algorithmic capability has been provided to select neighbors as those cells that share a common node (CNN) or those that share a common face (CFN). The ability to select CFN at interior cells and CNN at boundary cells has also been constructed. These options are not controllable by the user at this time. At the present time, only the CFN approach is used to construct neighborhood hierarchies. For each level, the cells that share a common face with the existing collection of cells are added to the collection, beginning with the central cell. If after $orderofaccuracy-1$ passes, the number of distinct cells in the neighborhood set of cells does not exceed the number of polynomial coefficients in the polynomial interpolation chosen, one more pass is performed to add cells to the neighborhood. If the number of cells is still not greater than the number of polynomial coefficients, the degree of the polynomial is lowered until this condition is met.

ENO Options

Three ENO interpolation options are provided at this time. The first is quite experimental and is based on the "best term" approach outlined in Section 4. The second and third options are based on the "best stencil" approach. In the second approach, the stencil is selected by comparing a single measure of variation evaluated for each polynomial in the neighborhood. In the third approach, a hierarchial selection process is used. A norm of the first variation (say sum of absolute values of the first derivatives) is compared for first-level neighbors of the cell under consideration. The stencil identifier is moved to the cell with a "substantially" lower norm. Next a norm of the second variation is compared for first-level neighbors of the current stencil identifier and the identifier is shifted to the cell with a substantially lower value of the norm. The process is repeated as many times as the degree of the interpolating polynomial selected. The interpolating polynomial corresponding to the final position of the stencil identifier is adjusted to match the original cell's cell-average value and

then used as the piecewise polynomial for that cell. The second method is the least computationally intensive, followed by the third which in turn is trailed by the first.

1-d, 2-d, Axisymmetric and 3-d Capability

The dimensionality of the problem can be selected as 1-d, 2-d or 3-d. This selection determines the type of polynomial expansion to be used (see the section on Spatial Accuracy). For 1-d problems, the hexahedral cell type is very convenient. For 2-d problems, the hexahedral or triangular prism cell types are useful. For 3-d problems, all cell types can be used. For axisymmetric problems, the 2-d polynomial can be selected along with an axisymmetric slice of a 3-d grid representing the problem. Fluxes in the third direction are not masked out and boundary conditions corresponding to no mass flux through those faces are used.

Integrated 2-d Plotting Capability

An integrated two-dimensional plotting capability is provided that is useful in applications requiring only one layer of cells of the hexahedral or triangular prism type. The display device is assumed to be a Tektronix 4100 or 4200 series color terminal. The computational grid can be displayed either in black and white or colored by a selected function value. Each 2-d cell face can be colored by a single color representing the cell-average value of the chosen function. Function profiles along a range of cells can be plotted. Monochrome or color line contours can be shown. One interesting contouring option available uses the local polynomial for each cell leading to possibly discontinuous contour lines at cell interfaces. The level of discontinuity can be a measure of local accuracy in smooth regions and can help identify flowfield discontinuities also.

Saving Least-Squares Polynomial Constructors

The least-squares polynomial evaluation procedure was described in Section 4 (Eq. 4.28). An option is provided to save the collection of neighbors of each cell along with the least-squares polynomial constructor for each cell given by $(\overline{A}^T\overline{A})^{-1}\overline{A}^T$ When the cell topology changes, the neighborhoods must be recomputed. Even if the

grid points move without a change in connectivity between cells, the least-squares constructor has to be reevaluated. These interrelationships are displayed in Fig. 13.1.

Structured to Unstructured Mesh Transformations

Single-zone structured grids can be given as input to the flow solver. The code can automatically convert this to produce an "unstructured" mesh with one of the three cell types. This is accomplished by using the cells as they are for the hexahedral cell type, by dividing each structured grid cell into two triangular prisms for that type, and by dividing it into five tetrahedra for the third type.

Save Metrics or Compute as Needed

The average values of cell-face metrics for each face can be stored if necessary or computed as needed.

Solution of Linear Equations

A set of linear equations must be solved for the least-squares reconstruction procedure. It has already been pointed out that the coefficient matrix is symmetric. It is also positive definite as long as the collection of neighborhood cells is suitable. For debugging purposes, the user can select an efficient linear-equation solver that is applicable only to symmetric positive-definite matrices or a general Gaussian elimination procedure with partial pivoting. The former employs no pivoting strategy.

Grid Smoothing Options

As described in Section 9 on grid generation, several grid smoothing procedures have been implemented and tested. These options will be documented in more detail in future editions because they are still being changed and developed substantially.

Grid Point Movement

Several grid-point movement strategies are being developed and tested in combination with various local grid editing features. These will be documented in future editions.

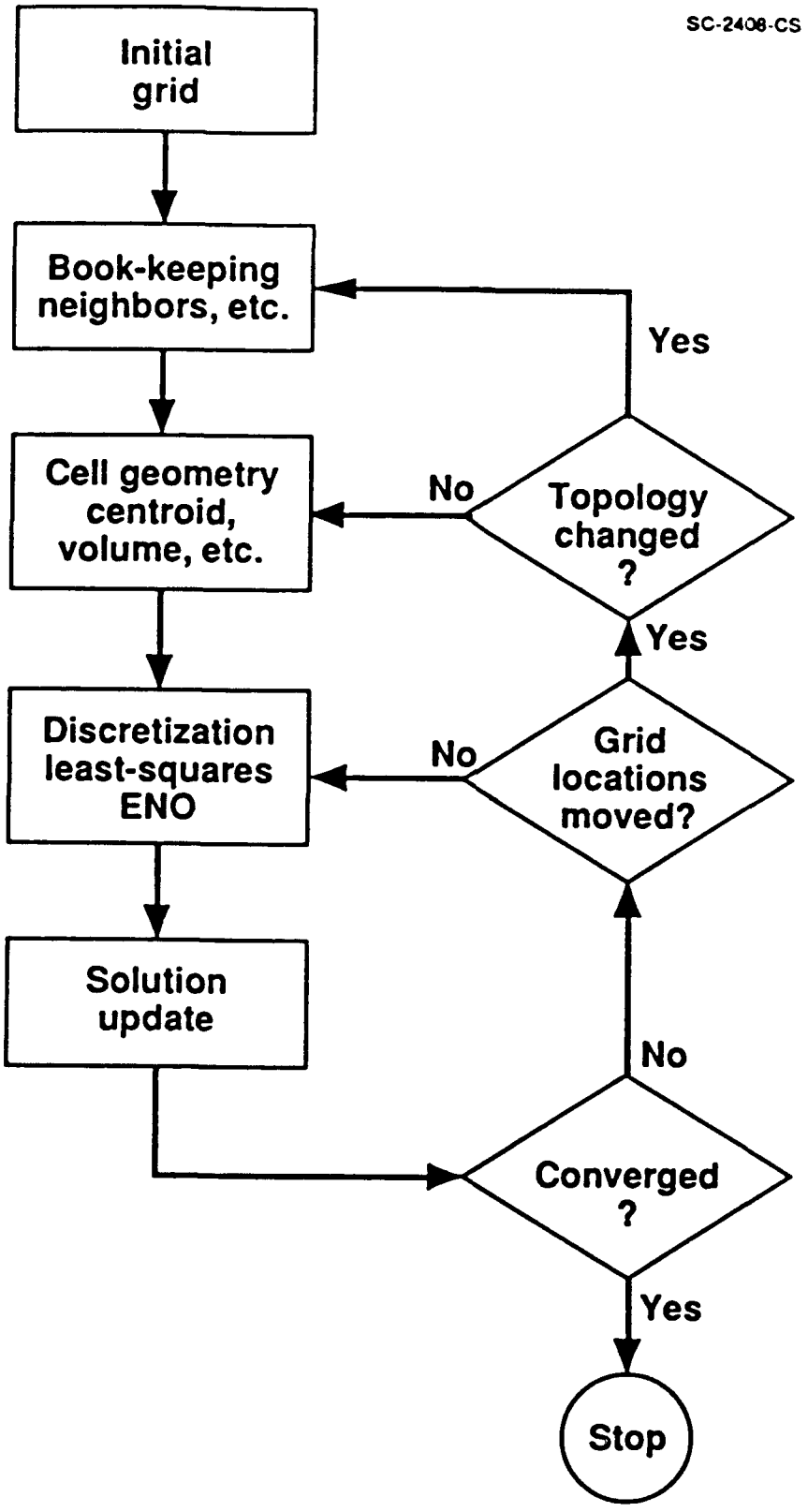


Figure 13.1 Top level flow chart for UNIVERSE-series codes

14.0 ILLUSTRATIVE EXAMPLES

Several results are presented in this section to provide an illustration of the computational capability of the UNIVERSE-series formulation presented in this report.

Linear Wave Equation

The one-dimensional linear wave equation was solved using fourth-order spatial and temporal accuracy with eighty hexahedral cells strung equally spaced along the x direction. The boundary condition was fixed at the left boundary and the initial conditions were chosen to be the step function. Figure 14.1 shows the profile of the dependent variable after enough time has elapsed to move the step function to the right by about half the computational domain. The third ENO formulation was employed. The lack of numerical oscillations near the step is proof that the ENO formulation is effective.

Burgers' Equation

The same case was repeated but this time by solving the inviscid Burgers' equation. The result is shown in in Fig. 14.2. The discontinuity is represented as a steep drop with no spurious numerical oscillations, once again verifying the ENO capability.

Ringleb Flow

The Ringleb flow[7] was studied using linear, quadratic, and cubic least-squares reconstruction. The expected order of accuracy is 2, 3, and 4, respectively. Both quadrilateral and triangular (prism) cells were employed. The latter are constructed by subdividing each quadrilateral cell. A fine grid with 700 quadrilateral cells (36 $\times$ 21 point structured mesh) and a coarse grid with 180 cells (19 $\times$ 11 point mesh) were used. Table 14.1 presents the results of the accuracy study undertaken using the two types of cells and two levels of grid resolution. The error is computed by measuring the l_1 norm of the difference between exact solution and numerical solution at the centroid of each cell. The four columns of numbers correspond to errors in

total energy per unit volume, density, x-momentum, and y-momentum, respectively. Results are presented for both midpoint and two-point Gaussian quadrature formulae. The results verify the accuracy of the implementation.

Subsonic Flow Past Circular Cylinder

An unstructured mesh with triangular-prism elements was employed for the inviscid flow past a circular cylinder (Fig. 14.3a). The pressure contours and surface pressure distribution obtained at a free stream Mach number of 0.4 is presented in Figs. 14.3b-c. Second-order least-squares reconstruction procedure was used here. The surface pressure distribution obtained using the USA-series code employing a second-order upwind TVD scheme (and quadrilateral cells) is also presented in Fig. 14.3c. This figure demonstrates the accuracy of the solution obtained using the least-squares reconstruction procedure.

Hypersonic Flow Past Circular Cylinder

Hypersonic flow past a circular cylinder ($M_\infty = 10$) was computed with third-order spatial accuracy and ENO option 1 using a structured mesh of 61×33 points. The built-in two-dimensional plotting capability was used to produce contour lines cell by cell using each cell's piecewise polynomial. As has been pointed out earlier, in high-gradient regions and near discontinuities, the left and right values at cell interfaces (based on the left and right polynomials) can be noticeably different. In smooth regions, the differences between left and right values will not be significant. This behavior is seen in Figure 14.4 which represents pressure contours. This example illustrates the capability of the formulation to capture very strong shock waves.

Sphere/Cylinder as Axisymmetric Flow

The ability to be able to compute axisymmetric flows is demonstrated in this next case where supersonic flow ($M_\infty = 3$) over a sphere/cylinder combination was computed using one slice of a spherical grid as has been described earlier. The grid is relatively coarse as can be seen from the figure. The same "discontinuous" contour plotting technique used in the previous case was employed for this case also and the

resulting contours are also seen in the figure. The pressure distribution along the surface of the sphere has also been presented.

G.R.I.D Configuration

Figure 14.6 shows a composite view of the unstructured 2-d grid with triangular elements and pressure contours for a calculation of a Mach 4 flow past a multiply connected region whose boundaries appear as the word "GRID". A single color is used in each grid element. The method used to generate the grid is based on a distance function concept presented in Ref. 39. First-order spatial accuracy was used to produce this result. This example serves to demonstrate the flexibility of the unstructured gridding approach and is otherwise not particularly significant.

Oblique Shock Problem

Figures 14.7a-d display results from a study of a two-dimensional problem in which there is a single oblique shock running from the lower left corner to the upper right corner of the grid shown in Fig. 14.7a. The free stream Mach number is 2.0 and the oblique shock angle is 40°. Free stream and after-shock conditions are imposed on the left and lower boundaries, respectively. There are 10 quadrilateral cells along each of the directions x and y. Pressure profiles plotted by connecting the cell-average values along x at various constant-y locations are shown in Figs. 14.7b-14.7d. Figure 14.7b was generated using first-order accurate reconstruction (which is automatically ENO). Figure 14.7c corresponds to second-order ENO reconstruction and Figure 14.7d results from fourth-order ENO reconstruction. These results serve to verify the effectiveness of the multidimensional ENO reconstruction.

For problems with steady shock waves, the best results can be obtained with a shock-aligned mesh and a good Riemann solver. In our flexible formulation, this can be achieved by just subdividing the quadrilateral cells that straddle the shock wave into two triangular cells. The corresponding grid and solution are shown in Figs. 14.7e-f. The Riemann solver due to Roe was used for this example.

Two-Dimensional "Aircraft" and "Store"

Figure 14.8 shows Gouraud-shaded pressure contours for a two-dimensional simulation of transonic flow ($M_\infty = 0.85, \alpha = 1.25°$) past a two-dimension projection of a fighter configuration provided by NWC.

F-18 Configuration

The surface grid and surface pressure contours from a computational simulation of transonic flow ($M_\infty = 0.85, \alpha = 0°$) past an F–18 configuration with pylon, rack, and bombs are displayed in Fig. 14.9.

Space Shuttle Multi-Body Configuration

Figures 14.10a-f present the geometry and corresponding flow-field solution for the Space Shuttle Orbiter with External Tank (ET) and Solid Rocket Booster (SRB) flying at $M_\infty = 1.55, \alpha = -5.5°$. The surface grid which is represented by 8731 nodes and 14386 triangular faces is shown in Fig. 14.10b. The solution is obtained using 18288 nodes and a total of 78737 tetrahedral cells. Figure 14.10d presents the surface pressure distribution on the fuselage at $\phi = 180°$ (upper center line). The chordwise pressure distributions on the upper surface of the wing are given in Fig. 14.10e. The results show that the present predictions are in good agreement with experimental data. The pressure distribution on the surface of the entire configuration is shown in Figs. 14.10c and 14.10f. The basic features of the flow (shocks, shock interaction between the Orbiter and ET, between the SRB and ET, and expansion waves) have been captured well. In contrast with earlier inviscid work,[11] the present approach required almost no effort for grid generation (the problem of geometry definition of the surface remains the same).

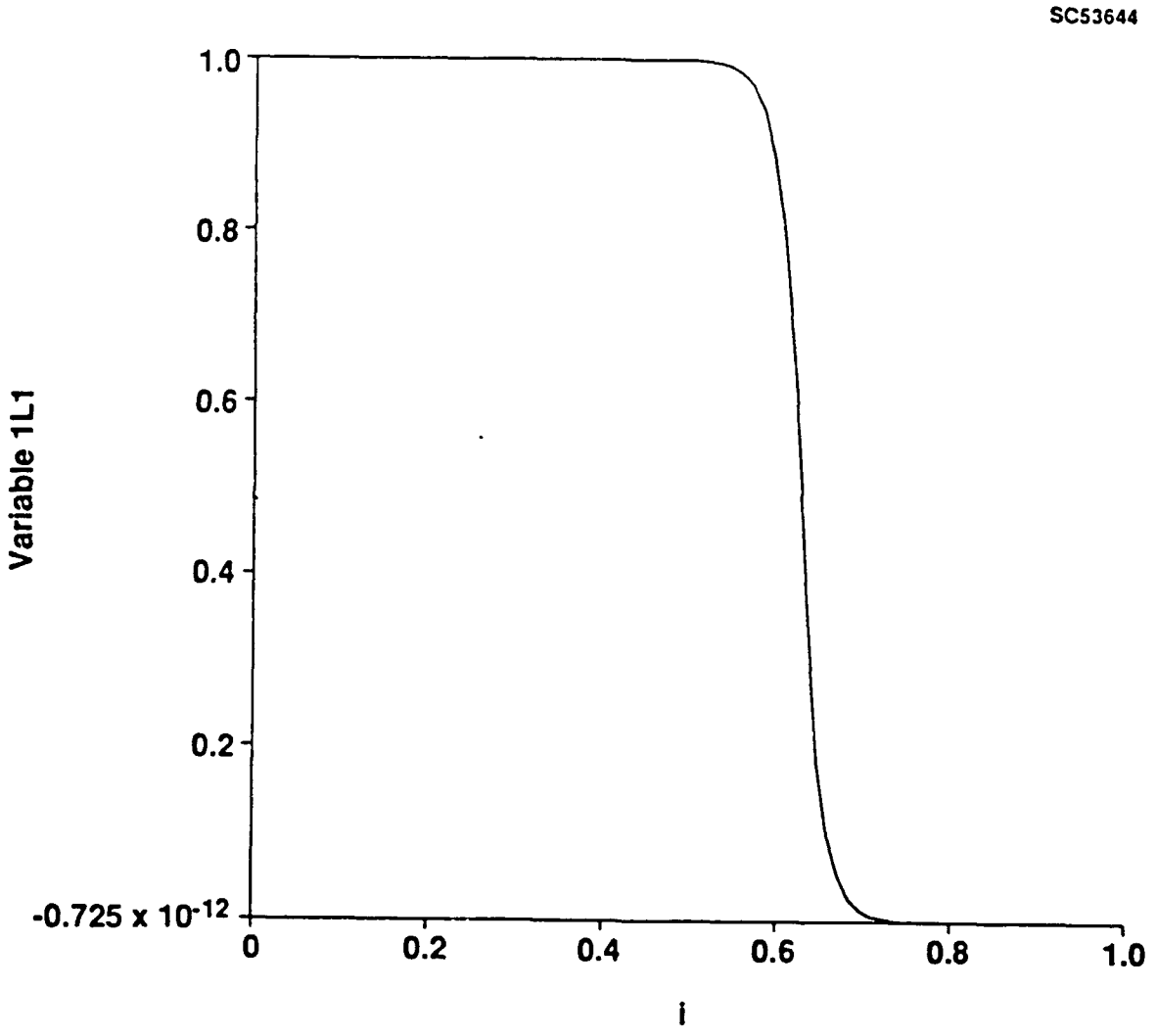


Figure 14.1 One-dimensional linear wave equation result

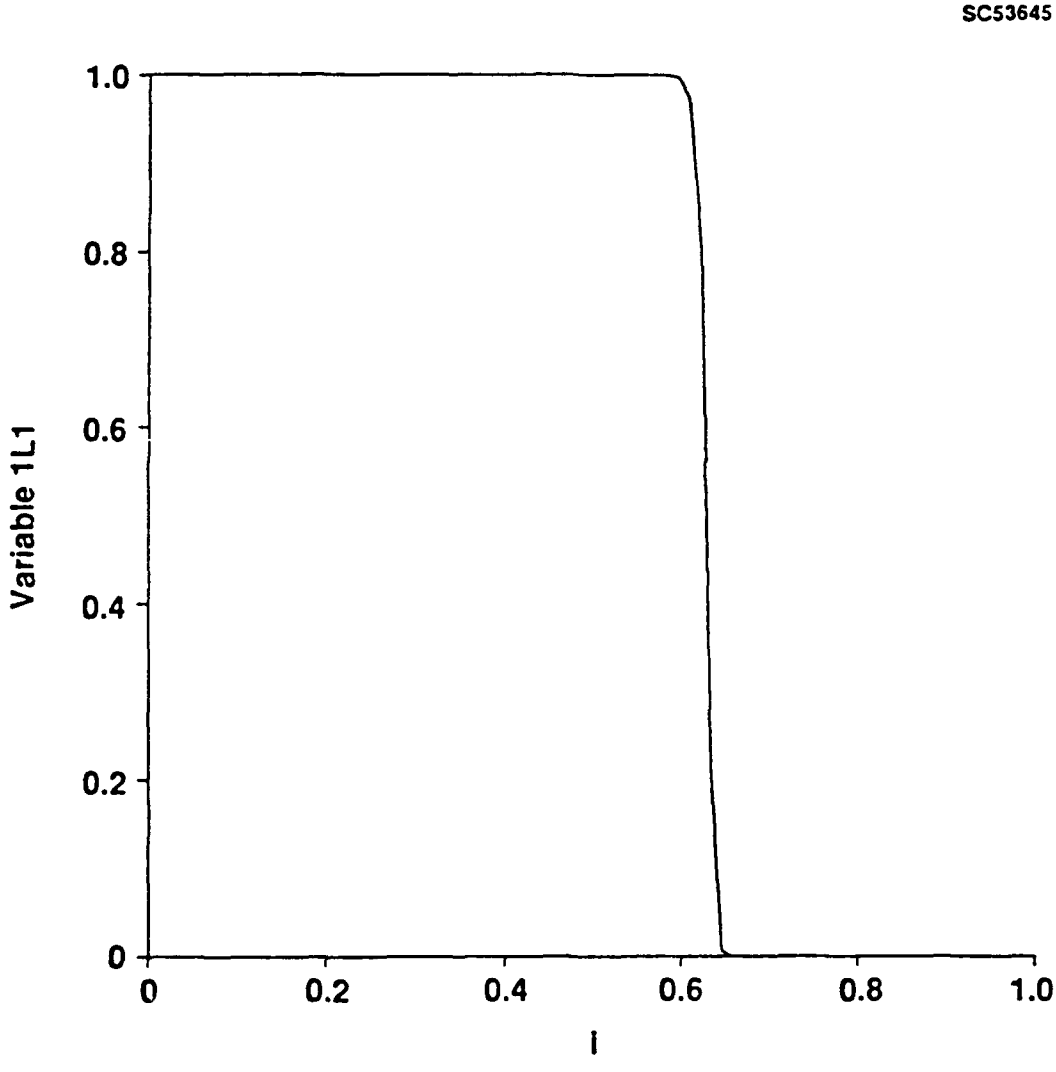


Figure 14.2 One-dimensional Burgers' equation result

```
FOR QUADRILATERAL GRID

2nd order
midpoint
COARSE GRID
3.377E-3  1.267E-3  1.765E-3  1.062E-3
FINE GRID
8.079E-4  3.000E-4  3.248E-4  2.174E-4
AVERAGE RATIO OF L1 NORM:  4.68

3rd order
midpoint
COARSE GRID
1.790E-3  6.481E-4  8.345E-4  4.969E-4
FINE GRID
3.431E-4  1.243E-4  1.558E-4  9.906E-5
AVERAGE RATIO OF L1 NORM:  5.20

3rd order
gaussian-two points
COARSE GRID
1.501E-3  5.902E-4  8.156E-4  5.173E-4
FINE GRID
1.939E-4  7.802E-5  1.122E-4  7.040E-5
AVERAGE RATIO OF L1 NORM:  7.48

4th order
gaussian-two points
COARSE GRID
1.394E-3  5.502E-4  5.616E-4  5.292E-4
FINE GRID
1.215E-4  4.829E-5  4.878E-5  3.091E-5
AVERAGE RATIO OF L1 NORM: 12.88


FOR TRIANGULAR GRID

3rd order
gaussian-two points
COARSE GRID
1.193E-3  4.376E-4  6.101E-4  3.892E-4
FINE   GRID
1.740E-4  6.312E-5  8.385E-5  4.776E-5
AVERAGE RATIO OF L1 NORM:  7.30

4th order
gaussian-two points
COARSE GRID
3.836E-4  1.456E-4  1.971E-4  1.566E-4
FINE GRID
2.777E-5  1.041E-5  1.329E-5  1.019E-5
AVERAGE RATIO OF L1 NORM: 14.50
```

Table 14.1 Ringleb flow results

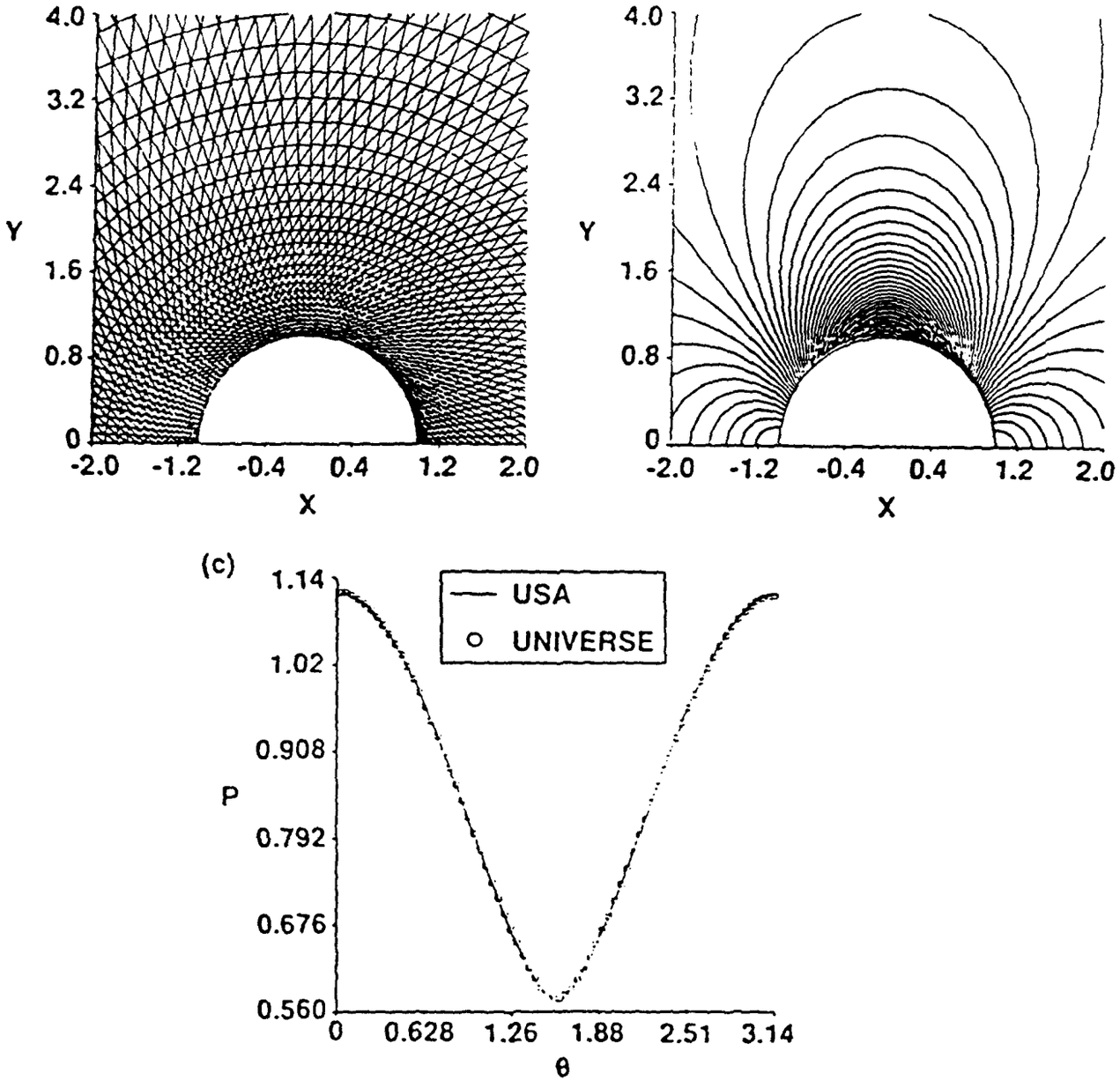


Figure 14.3 Subsonic flow past a circular cylinder

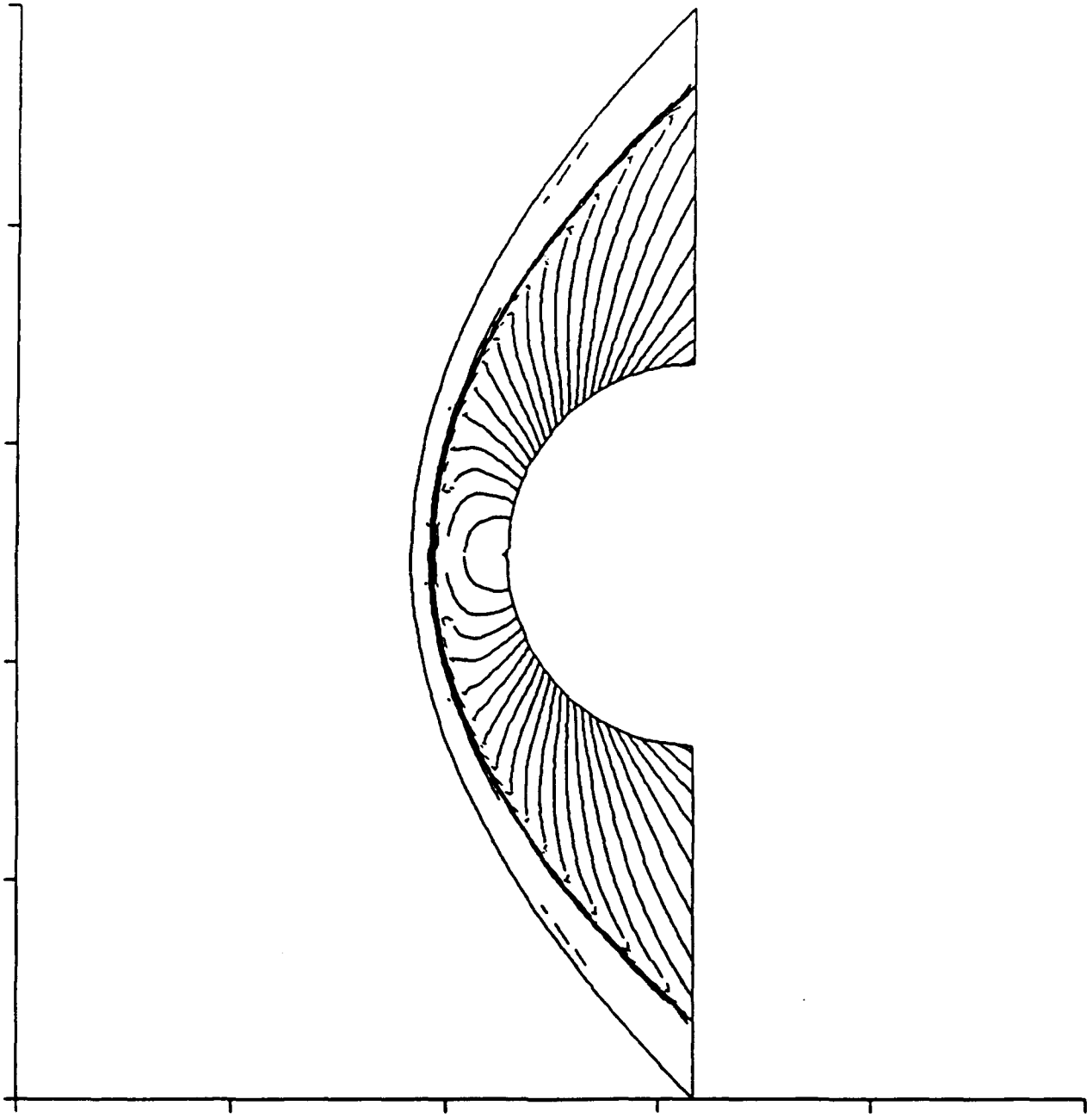


Figure 14.4 Hypersonic flow past a circular cylinder

SC53643

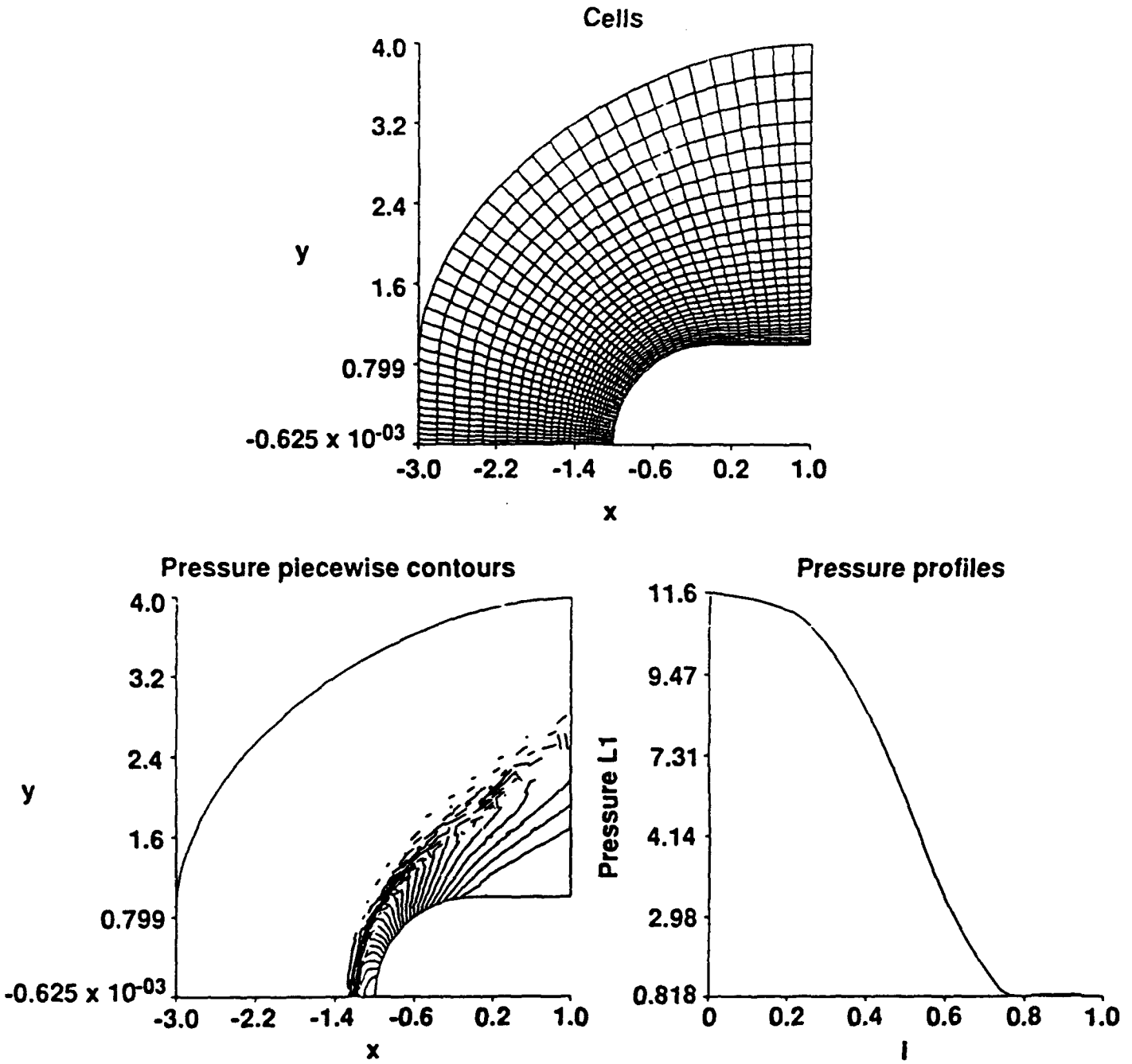


Figure 14.5 Supersonic flow past sphere/cylinder in axisymmetric mode

Figure 14.6 Flow past a "grid"

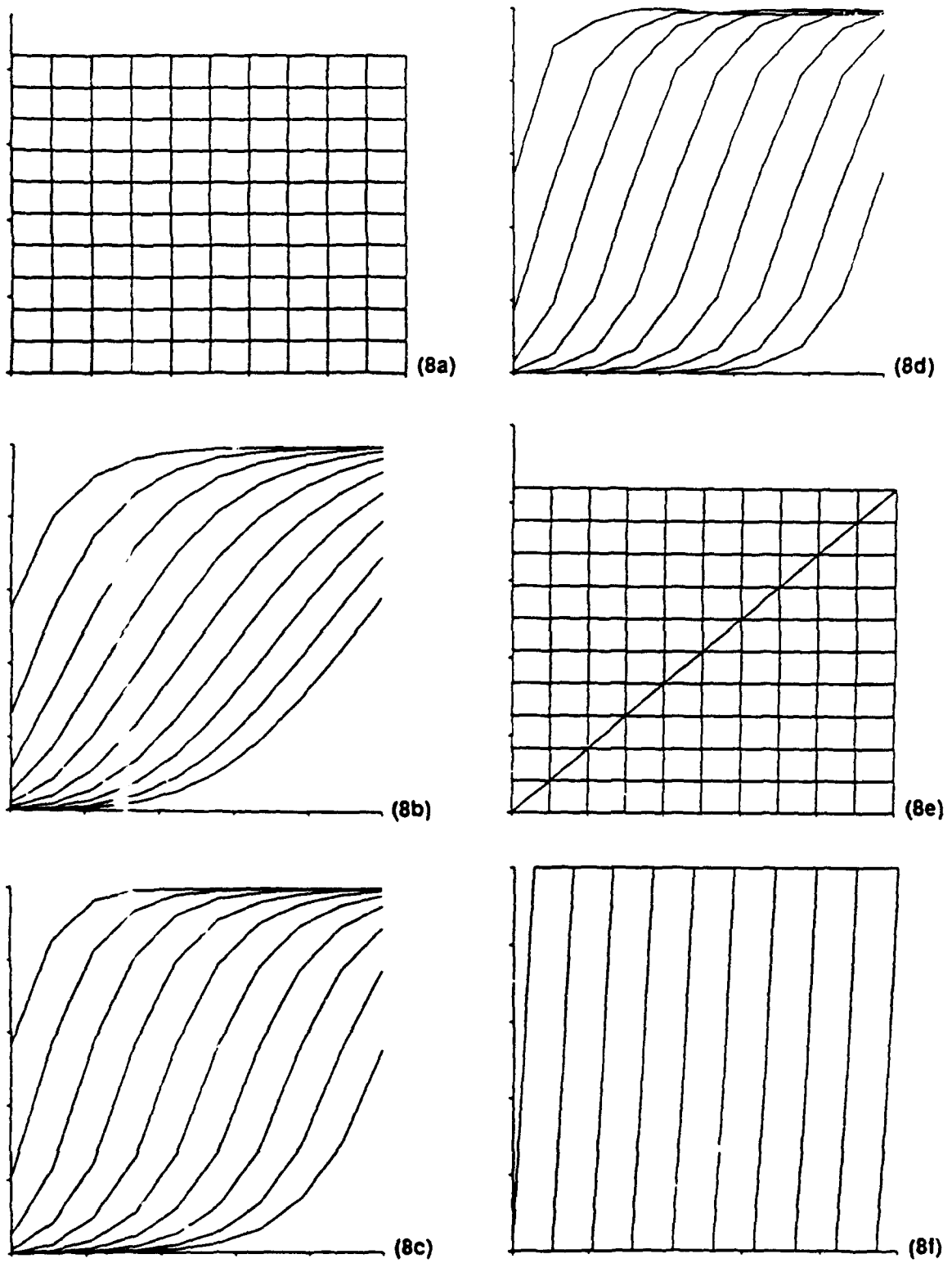


Figure 14.7 Oblique shock calculation

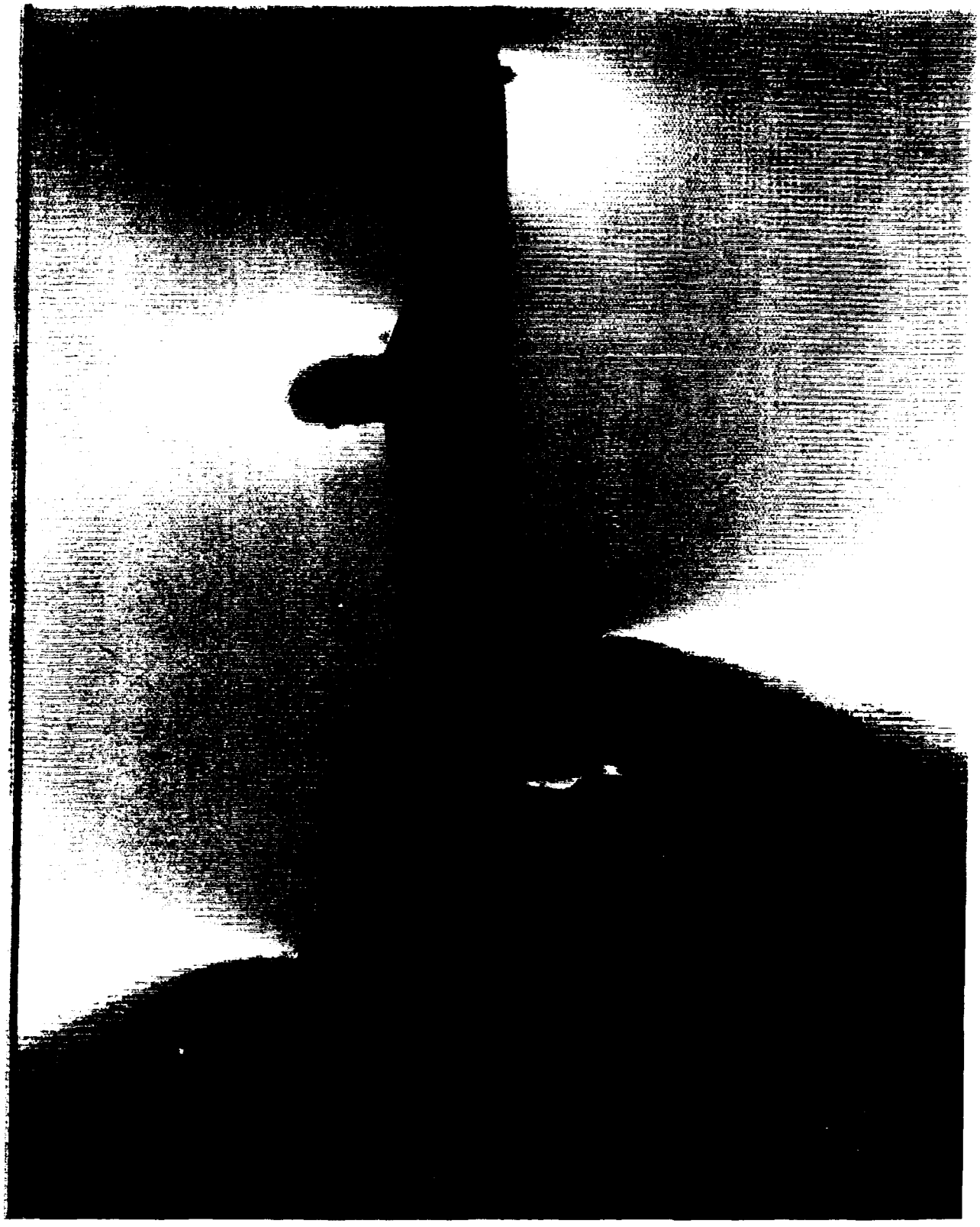


Figure 14.8 Simulation of a "two-dimensional" fighter

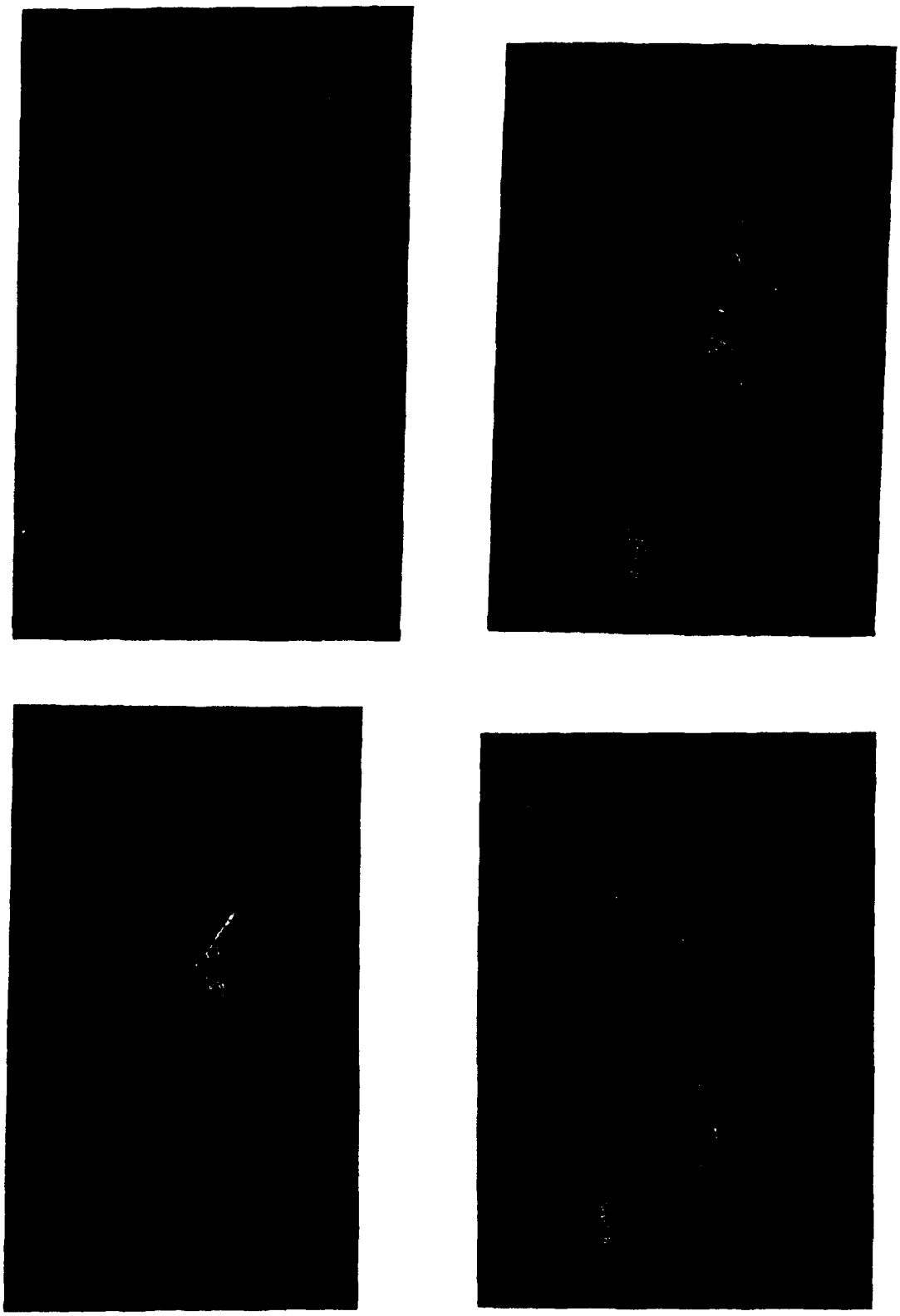


Figure 14.9 F–18 calculations

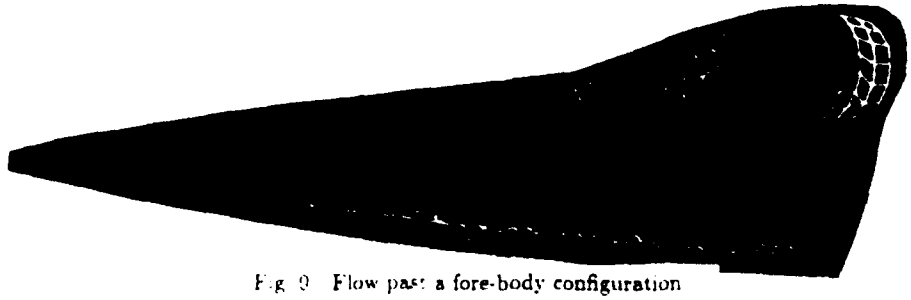
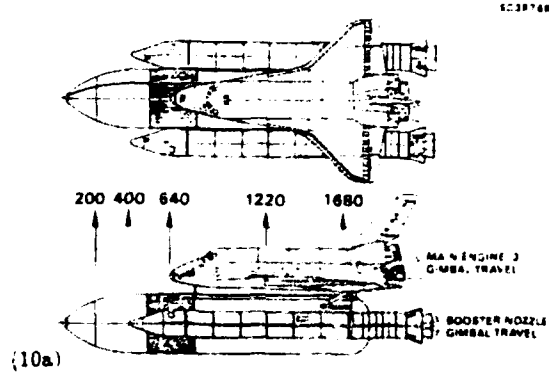
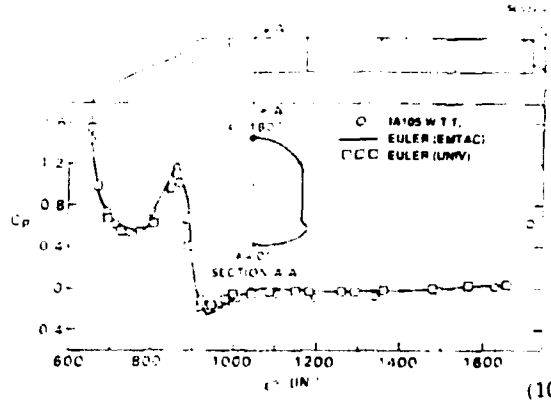


Fig. 9 Flow past a fore-body configuration

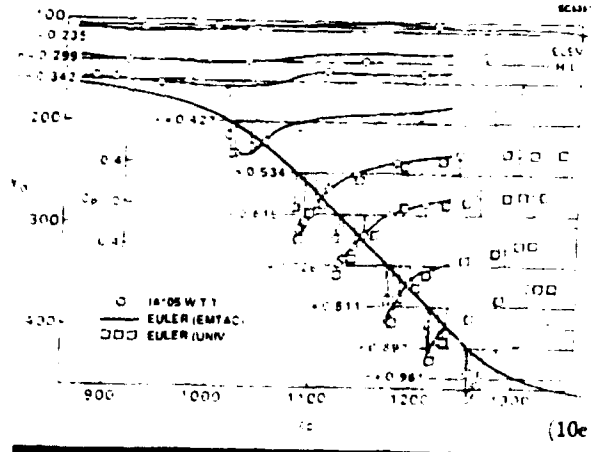


(10a)



(10d)

(10b)



(10e)

(10c)

(10f)

Figure 14.10 Space Shuttle calculations

15.0 REFERENCES

[1] A. Harten and S. Osher, "Uniformly High-Order Accurate Non-Oscillatory Schemes I," *SIAM Journal on Numerical Analysis*, Vol. 24, pp. 279-309, 1987; also MRC Technical Summary Report No. 2823, May 1985.

[2] A. Harten, B. Engquist, S. Osher and S. Chakravarthy, "Uniformly High Order Accurate Essentially Non-Oscillatory Schemes III," *Journal of Computational Physics*, Vol. 71, pp. 231-303, 1987; also ICASE Report No. 86-22, April 1986.

[3] S. R. Chakravarthy, A. Harten, and S. Osher, "Essentially Non-Oscillatory Shock-Capturing Schemes of Arbitrarily-High Accuracy," AIAA Paper 86-0339, January 1986.

[4] S. R. Chakravarthy, "Some Aspects of Essentially Nonoscillatory (ENO) Formulations for the Euler
Equations," NASA CR-4285, May 1990.

[5] J. Casper, "Finite-Volume Application of High Order ENO Schemes to Two-Dimensional Boundary-Value Problems," AIAA Paper 91-0631, January 1991.

[6] S.R. Chakravarthy, K.-Y. Szema and C.-L. Chen, "A Universe-Series Code for Inviscid CFD with Space Shuttle Applications Using Unstructured Grids," AIAA Paper 91-3340, September 1991.

[7] T.J. Barth and P.O. Frederickson, "Higher Order Solution of the Euler Equations on Unstructured Grids Using Quadratic Reconstruction," AIAA Paper 90-0013, January 1990.

[8] A. Harten and S.R. Chakravarthy, "Multi-Dimensional ENO Schemes for General Geometries," UCLA Computational and Applied Mathematics (CAM) Report 91-16.

[9] P. L. Roe, "Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes," *Journal of Computational Physics*, Vol. 43, 1981, pp. 357-372.

[10] P.L. Roe and J. Pike, "Efficient Construction and Utilization of Approximate Riemann Solutions," *Computing Methods in Applied Sciences and Engineering, VI,* Elsevier Science Publishers, INRIA, 1984.

[11] P.L. Roe, "Some Contributions to the Modelling of Discontinuous Flows," in: Large-Scale Computations in Fluid Mechanics, eds. B.E. Engquist, S. Osher and R.C.J. Somerville, *Lectures in Applied Mathematics*, Vol. 22, Part 1, 163-194, AMS, Providence, 1985.

[12] Roe, P.L., "Characteristic-Based Schemes for the Euler Equations," *Annual Review of Fluid Mechanics, 18*; 337-365, 1986.

[13] S. R. Chakravarthy and S. Osher, "Computing with High-resolu . Upwind Schemes for Hyperbolic Equations," *Lectures in Applied Mathematics, Volume 22*, Proceedings of the 1983 Summer Seminar on Large-Scale Computations in Fluid Mechanics, published by the American Mathematical Society.

[14] S.K. Godunov, "A Finite-Difference Method for the Numerical Computation of Discontinuous Solutions of the Equations of Fluid Dynamics," *Mat. Sb.*, Volume 47, 1959, 357-393.

[15] S. Osher and F. Solomon, "Upwind Schemes for Hyperbolic Systems of Conservation Laws," *Math. Comp.*, V. 38 (1982), pp. 339-377.

[16] S. R. Chakravarthy and S. Osher, "Numerical Experiments with the Osher Upwind Scheme for the Euler Equations", *AIAA Journal*, Vol. 21, No. 9, September 1983, pp. 1241-1248.

[17] S. R. Chakravarthy and S. Osher, "High Resolution Applications of the Osher Upwind Scheme for the Euler Equations," AIAA Paper 83-1943, Proceedings of the AIAA Sixth Computational Fluid Dynamics Conference, Danvers, Massachusetts, July 1983, pp. 363-372.

[18] A. Harten, P. Lax and B. Van Leer On Upstream Differencing and Gudunov-Type Schemes for Hyperbolic Conservation Laws *SIAM Review*, Vol. 25, 35-61 (1983); also ICASE Report No. 82-5 (March 1982).

[19] S. R. Chakravarthy, "Some New Approaches to Grid Generation, Discretization, and Solution Methods for CFD," presented at the 4-th International Symposium on Computational Fluid Dynamics held at the University of California, Davis, September 9-12, 1991.

[20] G.B. Whitham, *"Linear and Nonlinear Waves"*, John Wiley & Sons, 1974, pp. 39-40.

[21] K. Y. Szema, S. R. Chakravarthy, D. Pan, B. L. Bihari, W. T. Riba, V. M. Akdag, and H. S. Dresser, "The Application of a Unified Marching Technique for Flow Over Complex 3-Dimensional Configurations Across the Mach Number Range," AIAA Paper 88-0276, January 1988.

[22] S.R. Chakravarthy, K.-Y. Szema, and J.W. Haney, "Unified *Nose-to-Tail* Computational Method for Hypersonic Vehicle Applications," AIAA Paper 88-2564, June 1988;

[23] S.R. Chakravarthy and the Computational Fluid Dynamics Department, "Computational Aerodynamics Methodology for the Aerospace Plane," *Computing Systems in Engineering*, Vol. 1. Nos 2-4, pp. 415-435, 1990.

[24] D. Pan and S. Chakravarthy, "Unified Formulation for Incompressible Flows," AIAA Paper 89-0122, January 1989.

[25] D.K. Ota, S.R. Chakravarthy and J.C. Darling, "An Equilibrium Air Navier-Stokes Code for Hypersonic Flows," AIAA Paper 88-0419, January 1988.

[26] S. Palaniswamy, S.R. Chakravarthy, and D.K. Ota, "Finite-Rate Chemistry for USA-Series Codes: Formulation and Applications," AIAA Paper 89-0200, January 1989.

[27] S. Palaniswamy, D.K. Ota and S.R. Chakravarthy, "Some Reacting-Flow Validation Results for USA-Series Codes," AIAA Paper 91-0583, January 1991.

[28] U.C. Goldberg and S.R. Chakravarthy, "Prediction of Separated Flows with a New Backflow Turbulence Model," *AIAA Journal*, Volume 26, Number 4, April 1988, Pages 405-408.

[29] U.C. Goldberg and S.R. Chakravarthy, "Separated Flow Predictions Using a Hybrid k-L/Backflow Model," *AIAA Journal*, Volume 28, Number 6, June 1990, Pages 1005-1009.

[30] U. Goldberg and D. Ota, "A k-e Near-Wall Formulation for Separated Flows," AIAA Paper 90-1482, June 1990.

[31] U.C. Goldberg, "Turbulent Flow Computations With a Two-Time Scale One-Equation Model," AIAA Paper 91-1790, June 1991.

[32] S. Ramakrishnan and U. Goldberg, "Versatility of an Algebraic Backflow Turbulence Model," AIAA Paper 90-1485, June 1990.

[33] C.L. Chen, S. Ramakrishnan, K.Y. Szema, H.S. Dresser and K. Rajagopal, "Multi-Zonal Navier-Stokes Solutions for the Multi-Body Space Shuttle Configuration," AIAA Paper 90-0434, January 1990.

[34] C. L. Chen and S. R. Chakravarthy, "Calculation of Unsteady Rotor-stator Interaction," AIAA Paper 90-1544, June 1990.

[35] C.L. Chen and C.M. Hung, "Numerical Study of Juncture Flows," AIAA Paper 91-1660, June 1990.

[36] S. Osher, private communication.

[37] A. Harten, "ENO Schemes with Subcell Resolution," ICASE Report 87-56, NASA Langley Research Center, August 1987.

[38] S. Osher, "Riemann Solvers, the Entropy Condition, and Difference Approximations," *SIAM Journal on Numerical Analysis*, Vol. 21, No. 2, 1984, pp. 217-235.

[39] B. L. Bihari and S. R. Chakravarthy, "Unstructured Grid Generation Using the Distance Function," Proceedings of the Computational Fluid Dynamics Symposium on Aeropropulsion, held at NASA Lewis Research Center, Cleveland, Ohio, April 24-26, 1990.

[40] R. Löhner, "Some Useful Data Structures for the Generation of Unstructured Grids ," Communications in Applied Numerical Methods, Vol. 4, pp. 123-135, 1988.

[41] D. N. Knuth, "*The Art of Computer Programming*," Volume 3, Addison-Wesley, Reading, Massachusetts, 1973.

[42] J. Batina, "Unsteady Euler Algorithm with Unstructured Dynamic Mesh for Complex-Aircraft Aerodynamic Analysis," *AIAA Journal*, Vol. 29, No. 3, March 1991.

INITIAL DISTRIBUTION

16 Naval Air Systems Command
- AIR-526 (1)
- AIR-530 (1)
- AIR-5301 (1)
- AIR-53011D, Loezos (1)
- AIR-5302, Dudberly (1)
- AIR-53022, McMahon (1)
- AIR-53022B1, Mays (1)
- AIR-540 (1)
- AIR-540T (1)
- AIR-540TA (1)
- AIR-540TB (1)
- AIR-540TD (1)
- AIR-540TE (1)
- AIR-540TH (1)
- AIR-5404 (1)
- PMA-242 (1)

4 Chief of Naval Operations
- OP-502K (1)
- OP-506F (1)
- OP-911 (1)
- OP-982 (1)

8 Chief of Naval Research, Arlington
- OCNR-113 (1)
- OCNR-1132 (1)
- OCNR-12 (1)
- OCNR-121 (1)
- OCNR-20 (1)
- OCNR-21 (1)
- OCNR-213 (1)
- OCNR-22 (1)

7 Naval Sea Systems Command
- Cheng-T (1)
- SEA-06D (1)
- SEA-06U (1)
- SEA-62 (1)
- SEA-62D31 (1)
- SEA-63D (1)
- SEA-662 (1)

1 Commander-in-Chief, U. S. Pacific Fleet, Pearl Harbor (Code 325)
1 Headquarters, U. S. Marine Corps (MC-DLMW, Weapons Branch)
1 Commander, Seventh Fleet
1 Marine Corps Development Command, Quantico (Marine Corps Landing Force Development Center)
1 Naval Air Warfare Center Aircraft Division Warminster, Warminster (Code 604)
1 Naval Air Warfare Center Weapons Division, Point Mugu (Code 2061, Technical Library)
4 Naval Postgraduate School, Monterey
- Aeronautics Department, Code AA/HO, R. M. Howard (1)
- Dean of Research (1)
- Mechanical Engineering (1)
- Physics and Chemistry (1)

1 Naval Strike Warfare Center, Fallon (Intelligence Library)
1 Naval Surface Warfare Center, Indian Head Division, Indian Head (Code 26)

14 Naval Surface Warfare Center, Dahlgren Division, Dahlgren
- G06 (1)
- G07, F. Moore (1)
- G10 (1)
- G20 (2)
- G23
  - Haag (1)
  - Hardy (1)
  - Weisel (1)
- G30 (1)
- G42, Higdon (1)
- G43, Briggs (1)
- G103 (1)
- K23 (1)
- N12 (1)

4 Naval Surface Warfare Center, Dahlgren Division Detachment White Oak, Silver Spring
- G43 (1)
- K204 (1)
- R12 (1)
- R44 (1)

1 Naval War College, Newport
1 Naval Weapons Evaluation Facility, Kirtland Air Force Base
1 Naval Weapons Station, Yorktown (Code 47)
1 Army Missile Command, Redstone Arsenal (AMSMI-RD-ST, L. Mixon)
1 Army Materiel Systems Analysis Agency, Aberdeen Proving Ground
1 Harry Diamond Development Center, Adelphi (L. Cox)
2 Air Force Systems Command, Andrews Air Force Base
- AFSC/DLCA (1)
- AFSC/SDT (1)

1 Air Force Intelligence Agency, Bolling Air Force Base (AFIA/INT, Maj. R. Esaw)
7 Air Force Wright Laboratories, Armament Directorate, Eglin Air Force Base
- WL/ENMS, H. Sidhu (1)
- WL/MNA (1)
- WL/MNAG, Dr. Cloutier (1)
- WL/MNGA, L. Deibler (1)
- WL/MNMF (1)
- WL/MNP (1)
- WL/MNS (1)

1 Air Force Wright Laboratories, Wright-Patterson Air Force Base
- WL/FIMG (1)
- WL/FIMM (1)

1 Defense Advanced Research Projects Agency, Arlington (Strategic Technology Office)
2 Defense Technical Information Center, Alexandria
1 Deputy Under Secretary of Defense, Research and Advanced Technology (Engineering Technology)
1 Under Secretary of Defense (Office of Munitions)
1 Langley Research Center (NASA), Hampton, VA (J. M. Allen)
1 Boeing Aerospace and Electronics, Ridgecrest, CA (G. Ebling)
1 Fiber Materials, Incorporated, Biddeford, ME (J. Hueseman)
1 Hughes Aircraft Company, Ridgecrest, CA (C. E. Woodburn)
1 IIT Research Institute, Chicago, IL (Document Library, GACIAC)
1 Loral Aeronutronics, Ridgecrest, CA (Peterson)
1 Raytheon Company, Missile Systems Division, Bedford, MA (R. E. Peterson)
3 The Johns Hopkins University, Applied Physics Laboratory, Laurel, MD
- E. F. Lucero (1)
- E. T. Marley (1)
- L. B. Weckesser (1)